Theses and Dissertations   1. Thesis and Dissertation Collection, all items

1989-09

Implementation of ImageActionplus software for improved image analysis of solid propellant combustion holograms.

Hockgraver, Valerie Ruth

Monterey, California. Naval Postgraduate School

http://hdl.handle.net/10945/27089

NAVAL POSTGRADUATE SCHOOL
Monterey , California

THESIS

H64315

Implementation of ImageActionplus Software
for Image Analysis of Solid Propellant
Combustion Holograms

by

Valerie R. Hockgraver

September 1989

Thesis Advisor: John P. Powers

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE		Form Approved OMB No. 0704-0188

1a REPORT SECURITY CLASSIFICATION Unclassified	1b RESTRICTIVE MARKINGS

2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION / AVAILABILITY OF REPORT
2b DECLASSIFICATION / DOWNGRADING SCHEDULE	Approved for public release; distribution is unlimited

4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)

6a NAME OF PERFORMING ORGANIZATION Naval Postgraduate School	6b OFFICE SYMBOL (If applicable) 62	7a NAME OF MONITORING ORGANIZATION Naval Postgraduate School

6c ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000	7b ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000

8a NAME OF FUNDING / SPONSORING ORGANIZATION Air Force Astronautics Laboratory	8b OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER

8c ADDRESS (City, State, and ZIP Code) Edwards Air Force Base Edwards AFB, CA	10 SOURCE OF FUNDING NUMBERS			
	PROGRAM ELEMENT NO	PROJECT NO NOFO461 89-X-0006	TASK NO	WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification) Implementation of ImageActionplus Software for Improved Image Analysis of Solid Propellant Combustion Holograms

12 PERSONAL AUTHOR(S) HOCKGRAVER, Valerie R.

13a TYPE OF REPORT Master's Thesis	13b TIME COVERED FROM _____ TO _____	14 DATE OF REPORT (Year, Month, Day) September 1989	15 PAGE COUNT 91

16 SUPPLEMENTARY NOTATION

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number) Holograms; Image processing; Speckle; Data analysis; Combustion
FIELD	GROUP	SUB-GROUP	

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

This thesis supports computer-aided data analysis of holograms produced from rocket motor firings. The work reported in this thesis modified existing software code to make it compatible with installed upgrades in the microcomputer imaging system. In particular, this involved converting the format of C language function calls to ITEX/PC image processing software to that dictated by ITEX/PCplus software. Additional modifications were performed to enhance code portability and optimization. Results indicate that all efforts to incorporate the new system software have been successful.

20 DISTRIBUTION AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION Unclassified
22a NAME OF RESPONSIBLE INDIVIDUAL POWERS, John P.	22b TELEPHONE (Include Area Code) (408)646-2081	22c OFFICE SYMBOL 62Po

Implementation of ImageActionplus Software for Improved Image Analysis of Solid Propellant Combustion Holograms

by

Valerie Ruth Hockgraver
Lieutenant, United States Navy
B.S., Florida Southern College, 1981

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL

September 1989

ABSTRACT

This thesis supports computer-aided data analysis of holograms produced from rocket motor firings. The work reported in this thesis modified existing software code to make it compatible with installed upgrades in the microcomputer imaging system. In particular, this involved converting the format of C language function calls to ITEX/PC image processing software to that dictated by ITEX/PC*plus* software. Additional modifications were performed to enhance code portability and optimization. Results indicate that all efforts to incorporate the new system software have been successful.

C. I

TABLE OF CONTENTS

iv

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AFRT   Air Force Resolution Target

AVW   Active Video Window

DIR   Directory

DOS   Disk Operating System

EGA   Enhanced Graphics Adapter

IBM   International Business Machines

I/O   Input/Output

MSC   Microsoft C Compiler Ver. 5.0

PC   Personal Computer

SAR   Synthetic Aperture Radar

SI   Speckle Index

STATGRAPHICS   Statistical Graphics Systems

ACKNOWLEDGMENT

I. INTRODUCTION

A. BACKGROUND

This thesis was a continuation of the work currently being performed at the Naval Postgraduate School and was supported by the contributions attained in References 1, 2, 3 and 4. The goal of this research was to obtain data concerning the effect of propellant properties upon the performance of solid propellant rocket motors. Data analysis is performed on holographic images obtained from combustion products created during rocket firings. Holograms are obtained through the use of a pulsed ruby laser and a holocamera. Reconstruction of the holographic image for analysis is achieved with a krypton laser, variable-power microscope, and video camera. Unfortunately, the optical diffuser required to record the hologram creates a speckle-corrupted image which must be filtered before the data retrieval process is initiated. The data attained from these procedures is used in performance studies, stability analysis predictions, and exhaust signature diagnosis.

B. THESIS OBJECTIVES

The thesis objectives were:

1. To implement software modifications in existing local code in support of the new computer system configuration.

2. To enhance program code portability.

3. To optimize program code and attempt to reduce user interaction where possible.

4. To create a menu format for executable files.

All objectives were accomplished. ITEX/PC*plus* library routines have been successfully integrated into the image analysis programs. The program file *genfunc.c* was

1

used to optimize program size, and new functions were created for user convenience. A menu format was devised to make the image analysis procedure easier to use.

C. SOFTWARE SUPPORT

The associated system software consists of the following items:

- Image analysis programs created via thesis research in References 1, 2, 3, and 4,
- Imaging Technology's ImageAction*plus* Software package,
- Imaging Technology's ITEX/PC*plus* software library,
- IBM Disk Operating System version 3.3 (DOS),
- Microsoft C optimizing compiler version 5.0 (MSC), and
- Microsoft Codeview version 2.10.

The ImageAction*plus* software package is a menu-driven system that allows the user to process stored or live video images [Ref. 5]. The two-level menu organization provides image analysis, graphics, geometric operations, and image processing capabilities. This software package also has the ability to utilize user-generated special purpose script files. The script files are created in C language and are supported by libraries created by Imaging Technology, Inc. Imaging Technology's ITEX/PC*plus* software libraries are compatible only with the Microsoft PASCAL, FORTRAN, and C compilers. These libraries consist of image processing and graphics function subroutines used in conjunction with the PCVISION*plus* Frame Grabber board, a frame memory and video digitizer installed in the IBM PC/AT.

D. THESIS ORGANIZATION

This thesis is organized into five chapters. The introductory chapter provides the project background and goals. It also discusses thesis objectives and organization.

Chapter II describes the system hardware modifications. The PCVISION and the PCVISION*plus* hardware configurations are compared, and the respective hardware capabilities are presented. Also the hardware performance is briefly discussed.

The third chapter gives a synopsis of the image processing procedure and discusses actual software programs and improvements at length. The chapter also describes the techniques and software development tools utilized for program modification. Finally, Chapter III provides program specifics.

The software performance analysis is provided in Chapter IV. Program analysis consisted of comparisons in the following areas: executable program size, program execution times, and program result verification.

Lastly, Chapter V summarizes the findings obtained during the course of this research and provides recommendations for future thesis research.

II. SYSTEM HARDWARE MODIFICATIONS

A. HARDWARE CONFIGURATION

Digital image processing involves the achievement of the following objectives through a computer interface: recognition, segmentation, enhancement, or analysis of an image. The image processing and digitization system consists of the following components:

- An IBM PC/AT computer with 40 megabyte hard drive,
- An Intel INBOARD 386/AT microprocessor and 80387 math coprocessor,
- An AST ADVANTAGE memory expansion board,
- A PCVISION*plus* Frame Grabber board,
- A computer monitor with EGA display,
- A Panasonic color video data monitor, and
- A video cassette recorder.

A block diagram of the basic elements of the image processing system is presented in Figure 2.1. One of the primary components of this image processing system is the video cassette recorder, which provides the introduction of the image into the PCVISION*plus* Frame Grabber board. This image signal consists of timing and analog video information in a standard RS170 video format. Through the digitization process, the analog signal is converted to a digital format for processing and storage. The IBM PC/AT central processing unit accesses these digital values via the frame memory.

The PCVISION*plus* Frame Grabber is a single board that is connected directly into one of the 16-bit expansion slots of the IBM PC/AT computer. Pixels are stored in frame memory after the Frame Grabber digitizes the analog video signal at a

4
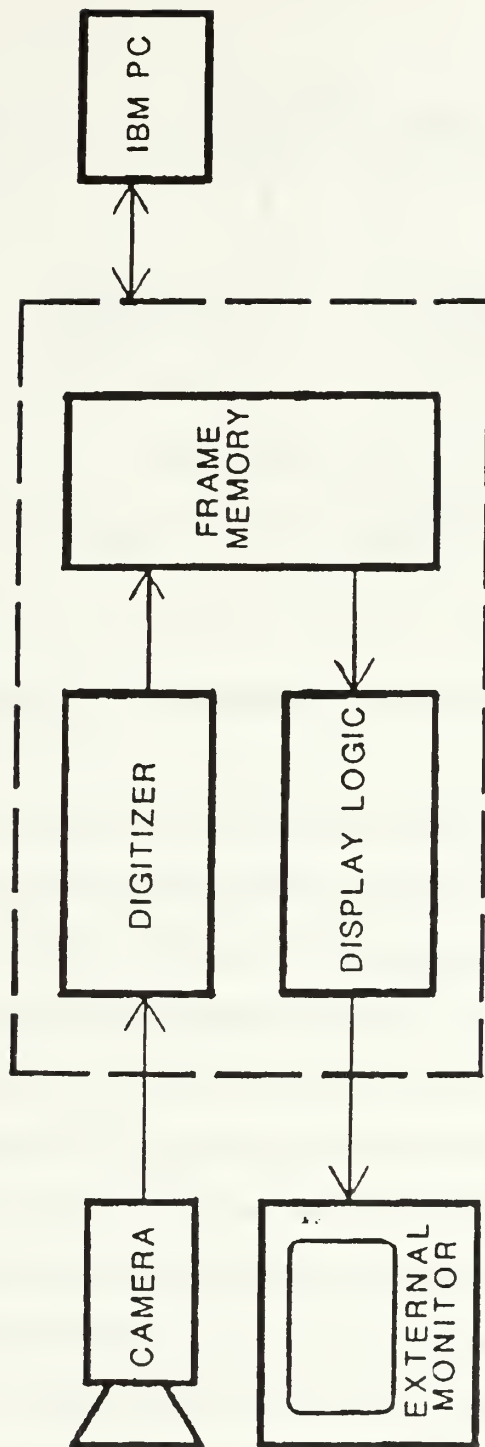
Figure 2.1: Elements of the Image Processing System. The Dotted Rectangle Encloses Frame Grabber Board. (From Ref. 7)

x,y=0,0 **511,0** **0,0** **511,0**

IMAGE AREA (A)   IMAGE AREA (B)

0,479 **511,479** **0,479** **511,479**

HIDDEN   AREA

0,511 **511,511**
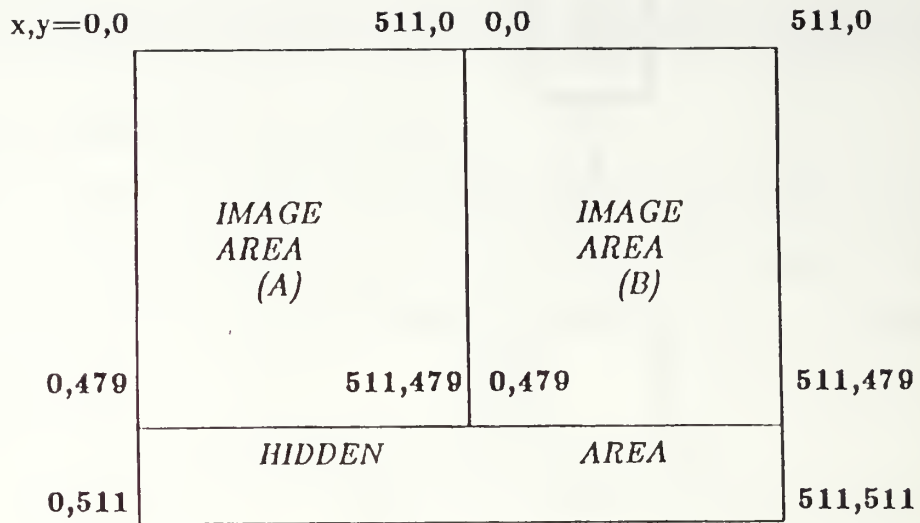
Figure 2.2: Display of AVW and Hidden Areas of Dual Memory Configuration. (From Ref. 6)

rate of thirty frames per second with eight bits of accuracy. In previous work, a PCVISION Frame Grabber board was utilized (versus the PCVISION*plus* version). The PCVISION*plus* board permits either two 512 x 512 pixel images or one 640 x 512 pixel image to be stored within frame memory [Ref. 7:p. 1-4]. In the PCVISION board, only one 512 x 512 pixel image can be stored [Ref. 8:p. 1-2]. The bottom thirty-two lines of the video output from both versions of the Frame Grabber board are within a "hidden area" which can be accessed by the host computer but not viewed on the video monitor without scrolling the image. The *active video window* (AVW) is the area within frame memory in which the pixel values are displayed or stored via the scanning circuit [Ref. 6: p. 1-14]. Figure 2.2 and Figure 2.3 indicate respectively the dual and single memory configuration hidden and AVW areas. Currently, only the dual memory configuration is utilized in thesis research.

x,y=0,0   640,0   1023,0

IMAGE AREA   HIDDEN AREA

0,479   640,479   1023,479

0,511   1023,511
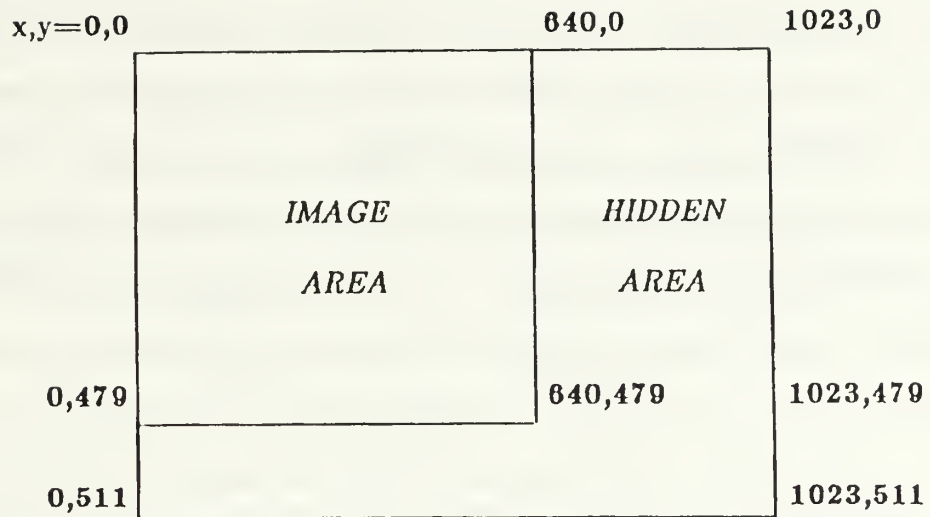
Figure 2.3: Display of AVW and Hidden Areas of Single Memory Configuration. (From Ref. 6)

Prior to display of the image on the video monitor, look-up tables (LUT) are employed for the transformation of pixel intensities. The original PCVISION Frame Grabber had only four LUTs associated with output [Ref. 8:p. 2-8]. The current Frame Grabber board has one input digitization path and three output channels, each allocated eight LUTs [Ref. 7:p. 1-4]. These LUTs allow for real time processing by virtue of their ability to afford simple point transformations of the 256 gray levels without any processing delay. Look up tables are addressed by eight bits of pixel information provided by the frame memory. A digital-to-analog converter and the look up tables constitute the display logic unit of both versions of the Frame Grabber boards. The digital-to-analog converter receives gray intensity values corresponding to the actual value addressed within the LUTs [Ref. 8:p. 2-9].

Both versions of the Frame Grabber board are compatible with the IBM PC/AT computer bus structure. This host interface provides the communication path between the Frame Grabber boards and the personal computer. The previous PC-VISION board host interface consisted of three components: the Interrupt Logic, Control Register Interface, and the Frame Memory Interface [Ref. 8:p. 2-1]. The PCVISION*plus* host interface is composed of the Control Register Interface and the Frame Memory Interface [Ref. 7:p. 4-2]. The Interrupt Logic had afforded the PCVISION Frame Grabber the ability to be controlled in real time while the IBM personal computer processed other instructions [Ref. 8:p. 2-3]. This outdated design feature has been superceded by transparent computer access to frame memory in the PCVISION*plus* system [Ref. 7:p. 1-6].

Within the input/output (I/O) space of the PC/AT, sixteen bytes are reserved for the PCVISION*plus* Frame Grabber control registers. Currently, only twelve of these sixteen bytes are being used by the system. Access to the remaining bytes is prohibited; they are reserved for future system growth [Ref. 7:p. 4-3]. On the contrary, the PCVISION board requires 32 bytes within the I/O space for its control registers. Only the first seven of these bytes are actually utilized for control functions [Ref. 8:p. 2-3].

The frame memory of the PCVISION*plus* system is apportioned into eight equivalent blocks of 64K bytes. Block dimensions are 512 pixels by 128 pixels, and the physical layout of the dual-store configuration frame memory is shown in Figure 2.4. This configuration can be contrasted to that of the 640x512 image in which memory is apportioned into eight blocks of 1024 pixels by 64 pixels. The single-store frame memory configuration can be viewed in Figure 2.5. Individual selection of these blocks is controlled by select bits within the Control Register. As viewed by the host, each block gives the semblance consecutive data bytes originating at the memory base

```
(0,0)                    (511,0)                  (1023,0)
      ┌─────────────────────┬─────────────────────┐
      │      Block 0        │      Block 4        │
(0,128)├─────────────────────┼─────────────────────┤(1023,128)
      │      Block 1        │      Block 5        │
      ├─────────────────────┼─────────────────────┤
      │      Block 2        │      Block 6        │
      ├─────────────────────┼─────────────────────┤
      │      Block 3        │      Block 7        │
(0,511)└─────────────────────┴─────────────────────┘(1023,511)
```
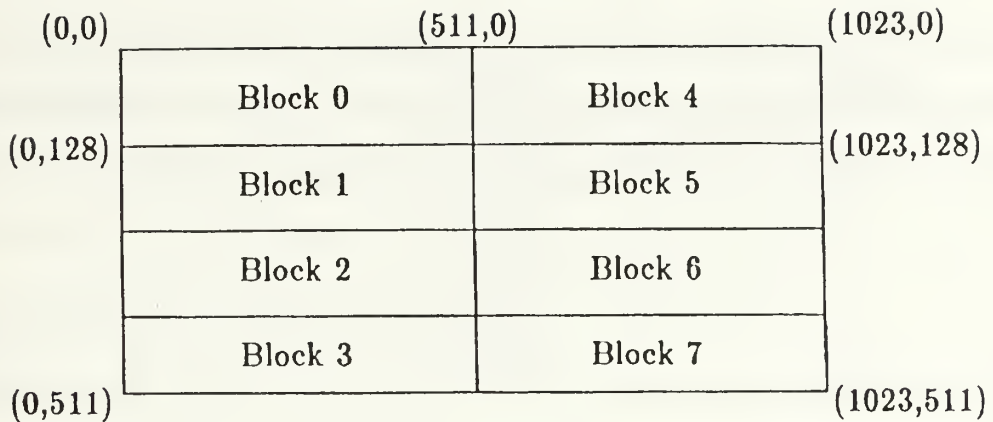
Figure 2.4: PCVISION*plus* **Dual Frame Memory Layout. (From Ref. 7)**

address [Ref. 7:p.4-3]. For the PCVISION system, the frame memory was subdivided into four blocks of 256 pixels by 256 pixels, forming a quadrant structure as displayed in Figure 2.6. At any time only one block can be accessed from the personal computer bus [Ref. 8:p. 2-7].

Both versions of the Frame Grabber board require the assignment of a single block of 64K bytes within the IBM PC/AT memory space located above 640K. Specialized software controls the mapping of individual blocks of frame memory into this reserved space [Refs. 7 and 8]. In the case of the PCVISION*plus* board, the memory base address is factory-configured at address A0000H. This address may be modified upon board installation provided the subject address selected is within the guidelines set forth in Reference 7.

It was originally intended that this thesis work would be implemented on a COMPAQ DESKPRO 386/20 computer system. The PCVISION*plus* memory base

(0,0)   (1023,0)

(0,64)   Block 0   (1023,64)
              Block 1
              Block 2
              Block 3
              Block 4
              Block 5
              Block 6
(0,511)   Block 7   (1023,511)
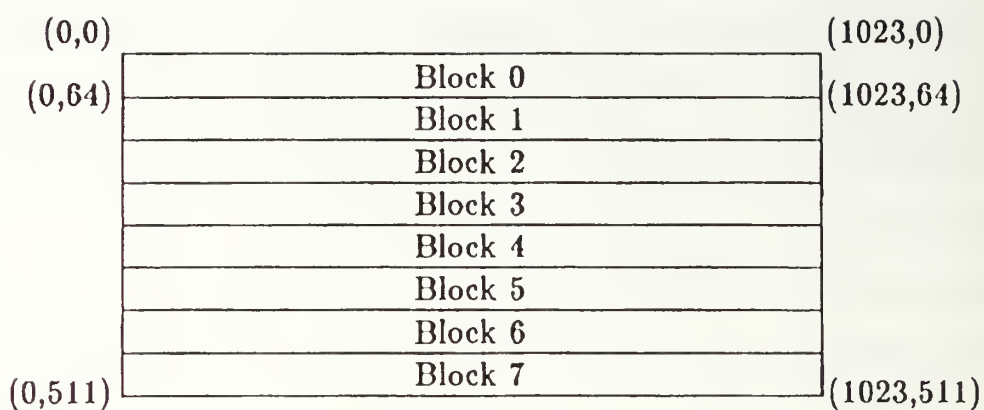
Figure 2.5: PCVISION*plus* Single Frame Memory Layout. (From Ref. 7)
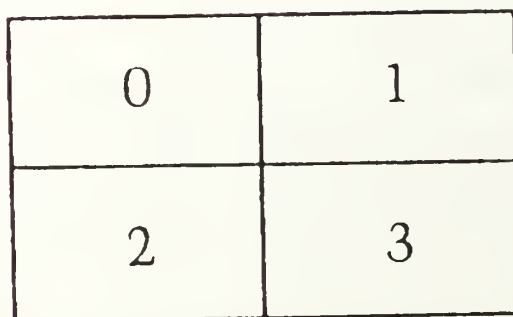
Figure 2.6: PCVISION Frame Memory Layout. (From Ref. 8)

address was configured to an approved value of D0000H. Therefore, the address region from D0000H to DFFFFH was reserved within the computer memory space, and all other memory and peripherals were restricted access to that memory region. Upon subsequent testing and diagnostic checks, it was found that there was an unresolved memory conflict within the COMPAQ computer memory which affected some of the operations of the Frame Grabber board. Due to necessity, the board was consequently installed in the IBM PC/AT. Memory base address D0000H was again selected; this time there were no resident memory conflicts.

B. HARDWARE PERFORMANCE

With the introduction of the PCVISION*plus* Frame Grabber board, the ability to process two separate images in frame memory is invoked. The dual memory configuration allows the host to select either image area for analysis and manipulation. A display memory function allows the user to view independently either image area. These capabilities allow for considerable future flexibility in the analysis of the rocket motor holograms.

The PCVISION*plus* board not only provides more image processing capabilities than the PCVISION board, but it also establishes the need for additional function libraries to support its abilities. For this reason, image analysis programs devised in prior thesis research had to be modified to reflect the new supporting libraries. These programs and their resulting modifications are emphasized in the following chapter.

III. SOFTWARE PROGRAMS AND IMPROVEMENTS

A. IMAGE PROCESSING PROCEDURE

As was discussed in the introductory chapter, holographic techniques are employed to obtain the image of combustion products from small rocket motors during firing. During the hologram reconstruction process, a krypton laser is used in conjunction with a variable power microscope. Due to the presence of a diffuser that is required to eliminate thermal gradient effects, speckle noise is introduced into the captured image. The following paragraphs describe the procedure developed to acquire and process this image for statistical study.

1. Image Acquisition

The speckle-corrupted image is recorded for future processing via a combination of a video cassette recorder and low-light-level camera attached to the variable power microscope. The magnification power of the microscope determines the relative size of the pixel elements comprising the image. Pixels are the standard unit for the measurement of the features within the image. Object sizes are determined by scaling the measured object width (in pixels) by the dimension spanned by one pixel (in microns per pixel). A quantization error is introduced into this calculation due to the necessity of measuring dimensions with an integer number of pixels. As the width decreases, the proportional level of uncertainty increases, which produces a more erroneous scale factor calculation. A calibration object is inserted within the holographic image to determine the width spanned by a pixel. It is essential that this object covers an appreciable portion of the monitor screen (at least one half of the

12

area) to counteract this quantization error. The calculation of the conversion factor from pixels to microns becomes increasingly important in subsequent steps of image processing.

2. Image Digitization

From the VCR tape obtained from the hologram reconstruction, the menu-driven ImageAction*plus* software is used to "grab" and digitize the desired image. The PCVISION*plus* Frame Grabber board implements this procedure. The digitized video image is comprised of a 512 x 480 array of pixels having gray intensity levels ranging from 0 to 255. These intensity levels extend from blackest-black, level 0, to whitest-white, level 255.

3. Speckle Reduction Filtering

Image filtering is the paramount technique used in speckle reduction. The amount of speckle reduction achieved by filtering algorithms is gauged by a calculation of the speckle index. Speckle index is defined as the ratio of the image's local deviation to the local mean of pixel values, and this measure is indicative of the random speckle noise inherent in the image [Ref. 1:pp. 17–19].

The three filtering algorithms are based upon synthetic aperture radar (SAR) theory presented in References 9 and 10. There are three separate filter models. Derived from nonlinear techniques, two filter designs are based upon statistical methods. The third filter model has its foundation in a geometric hulling algorithm. The geometric hulling algorithm produces the best results by enabling the user to discontinue filtering at a less severe level of resolution degradation.

4. Image Threshold

In an ideal image, feature particles and the background would have distinct gray intensity levels. The process of thresholding creates a binary image in which

image features will appear black and the background white. In reality, the thresholded image will contain some amount of speckle manifested as feature particles and will have lost portions of actual particles during the filtering and thresholding process. Image evaluation errors are inherent whenever the grey levels of the object overlap the gray levels of the image background. The selection of a threshold value is obtained from observation of the filtered image. ITEX/PC*plus* software allows the user to select a threshold value for which the designated LUT is modified. This modification is not made permanent until the image is mapped into frame memory, allowing the user to rapidly iterate values to make a determination on the most appropriate threshold value.

5. Feature Identification

Once the filtered image has been thresholded, it is ready to undergo feature identification. It is imperative that the programmed image has been previously thresholded due to the logic of the identification algorithm. This procedure locates all connected adjacent feature pixels and assigns feature value numbers to them. Through this labeling process, interconnected pixels are identified as an individual object. The total number of objects recognized serves as input for the next step of image processing. This number is restricted to 32,767 or below based upon the limitations imposed by the C language on integer values [Ref. 4:p. 9].

6. Feature Sizing

The procedure of feature sizing determines both the maximum vertical and horizontal diameters of the identified objects. The total area of the subject particles is computed, and all these measurements are delineated in terms of microns. As described before, a conversion factor is necessary for manipulation of the feature particle's pixel measurements into actual physical dimensions. This constant, called SCALE_FACTOR, is presently based upon the dimensions of a threaded screw at the

14

same magnification of the captured image. The screw threads serve as a calibration object to transform image object pixel size data into actual physical measurement criteria. The final output of the feature sizing routine is presented in tabular form suitable for input into the STATGRAPHICS software package, a statistical analysis tool.

7. Histogram production

To exhibit feature size data in a histogram form for ready evaluation, the STATGRAPHICS software package is utilized. This particular software package was selected because it is compatible with the IBM-PC/AT system and is relatively easy to use. Histograms produced by STATGRAPHICS are used for further statistical analysis to determine total particle distributions of holograms.

B. PROGRAM OPTIMIZATION

The primary purpose of this thesis research was to modify existing C language routines to make them compatible with the ITEX/PC*plus* software. Subsequent program optimization was performed in support of this goal. One significant improvement was the removal of all general-use functions from the main programs. Although a general support functions program file had been created in the previous rendition, this file had not been adequately utilized. As a result, all the main programs had needlessly large file sizes.

The C language provides the ability to use the infinitely-abusable GO TO statement. This creates a situation in which branching occurs in a deeply nested structure. The GO TO organization may prove effective in some instances, but it always produces confusion for the individual having the misfortune to try to modify that code sequence. For this reason, program structures were modified to eliminate all GO TO branch statements, thereby making code more modular and portable.

A menu format was developed for use with the executable programs. This format will facilitate demonstrations of image processing and makes the routines "user-friendly". In addition, supporting functions were devised for some of the programs to further eliminate the need for user interaction with the programs once the selected image had been loaded and image processing regions were identified. These modifications facilitated more rapid program execution. A more detailed discussion of program modification specifics will occur in a succeeding section.

C. SOFTWARE DEVELOPMENT TOOLS

The Microsoft Codeview Debugger program was utilized for program development. It requires that specific compiler directives be used during program compilation. Although Codeview is an excellent developmental tool, program debugging can become a tedious process. No program modifications can be performed while in Codeview, and for that reason, it is not an adequate tool for what is needed for program modules of this size. The ITEX/PC*plus* routines require large memory module handlers, and few other debuggers are able to meet this requirement. Quick C version 2.0 is capable of handling large memory modules and has an online source debugger which is more versatile than Codeview [Ref. 11]. It is recommended that in future work with these program modules, Quick C should be included in system upgrades.

Several batch files were created for compilation and linker processes since Codeview required special compiler directives. There are two different versions of these files: compilation and link with debugger information, and compilation without debugger information for file optimization and compaction. Listings and justifications for these DOS Batch files are referenced in Appendix K.

D. PROGRAM SPECIFICS

The succeeding paragraphs discuss the programs that were developed in support of this thesis research. These existing programs were modified to be compatible with the ITEX/PC*plus* software libraries and the ImageAction*plus* routines. Where at all possible, program optimization was performed. In some cases, code had to be completely revamped to accommodate new requirements. All source code listings are included as appendices at the end of this document. A block flow diagram of program usage is provided in Figure 3.1.

1. Program File *thesis.h*

The program header file *thesis.h* (Appendix A) contains all requisite library include files, C manifest constants, and MACRO definitions as necessitated by main programs. For code portability, it is imperative that a centralized copy of all common declarations and definitions be maintained as the programs evolve. Therefore, any modifications to common parameters can be performed by substitutions within this file.

With the advent of ITEX/PC*plus* software, additional include files were necessary beyond the scope of the original header file listed in Reference 4. For example, ITEX/PC libraries require the use of the include file *itexpc.h* [Ref. 12:p. 1-15]. This requirement has been superceded by the ITEX/PC*plus* software which dictates the use of *itexpfg.h* and *stdtyp.h* include files. Not only must these include files be listed in the header file, but to be utilized, they must also be resident in the actual program directory [Ref. 6:p. 1-9].

The added capability of dual memory required that many of the calls to the software libraries be altered. Additional manifest constants were needed for memory specification. They included MEMORY, defining the type of frame memory configuration, and constants necessary to define the size of the selected frame memory

17
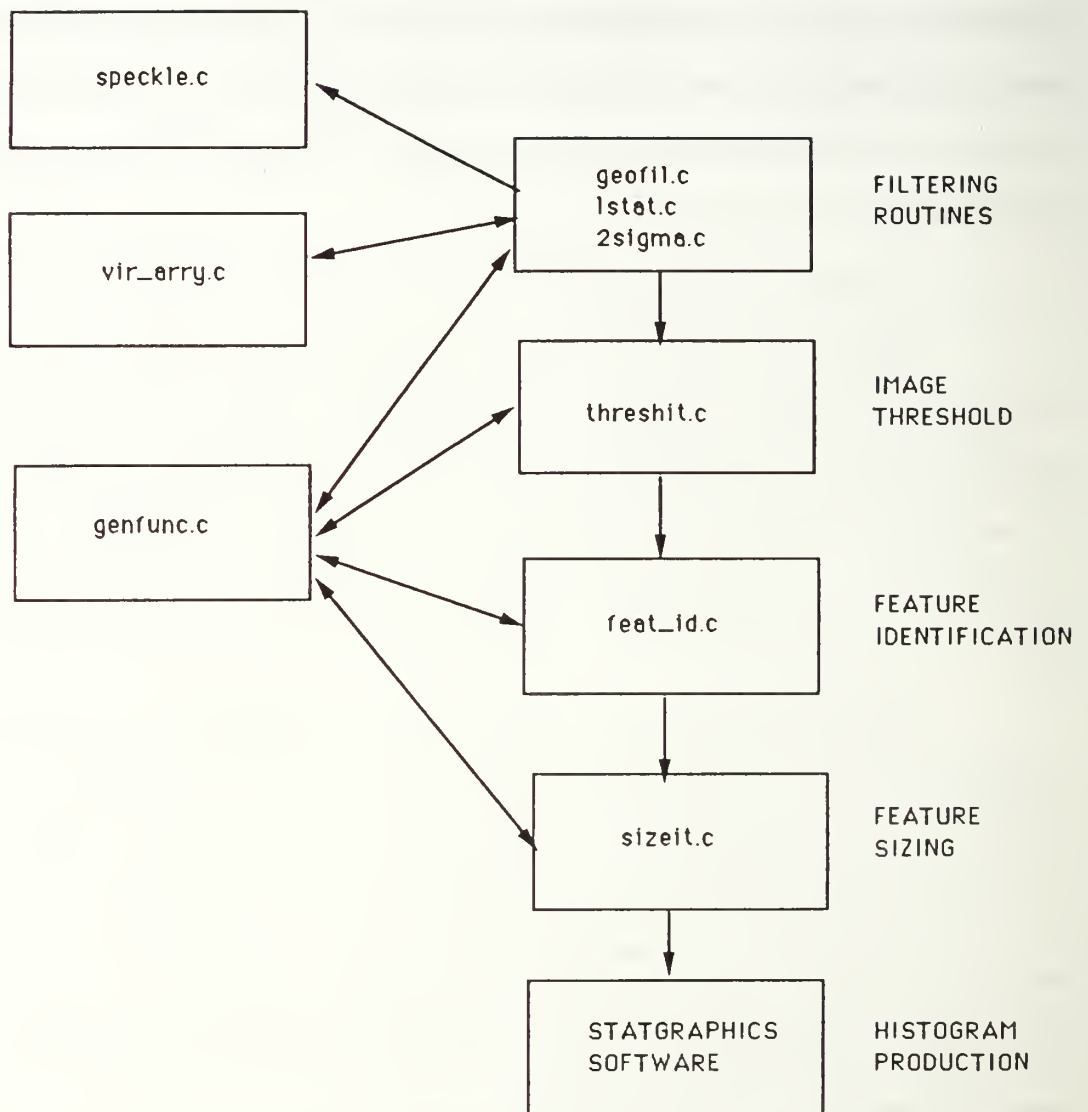
Figure 3.1: Flow Diagram of Image Processing Programs.

configuration: XSIZE, YSIZE, and DEPTH. XSIZE and YSIZE indicate respectively the horizontal and vertical coordinates of the memory size, and DEPTH refers to the number of bits assigned per pixel in frame memory [Ref. 6:p. 2-6].

2. Program File *geofil.c*

Appendix B contains the listing for *geofil.c*, a program based upon a geometric hulling algorithm introduced in References 9 and 10. The algorithm is applied to the image and its complement by performing comparisons between a central pixel value and adjacent pixels in horizontal, vertical and diagonal directions. The original geometric filter routines were adapted by Edwards [Ref. 1] from information set forth in References 9 and 10; and Kaeser successfully adapted these to the C language [Ref. 4]. Edwards' FORTRAN filtering routine was capable of handling only a quarter of the actual image due to memory restrictions, whereas the C language version fully supported the entire image. A large part of the C version's success was due to the utilization of virtual memory arrays for data processing. These arrays are stored on disk, but they are accessed as though they are stored in memory. File constraints within the associated operating system pose the only limitation upon the virtual array size [Ref. 13].

Few modifications were performed on the *geofil.c* program to make it compatible with the ITEX/PC*plus* software. The majority of the changes for compatibility encompassed general support functions. However, much of the C language code was altered to attain optimum modularity. This filter program had incorporated the use of GO TO statements to facilitate branching between the filter stepping functions. Alterations included the removal of these subject statements and the formation of two recursive functions. These recursive functions support program modularity and permit easier modifications for future updates.

One common modification performed on *geofil.c* and its other main program counterparts involved the declaration of the image filename and comment line (comline). The header file used by ITEX/PC, *itexpc.h*, required special file type structures used in conjunction with parameter passing for SAVIM and READIM functions. These structures consisted of strings with length 20 or 200 characters, and were classified as typedef LS20 or LS200, respectively [Ref. 12:p. 1-19]. ITEX/PC*plus* limited filename to 20 characters and comment lines to 200 characters, but removed the special structure requirement. SAVIM and READIM routines now require that the image filename and comment line be classified as pointers of type character [Ref. 6]. Consequently, all program files were altered to reflect this requirement.

3. Program File *lstat.c*

The program *lstat.c* (Appendix C) uses local statistical methods to calculate a weight, k, which determines the gray level of a pixel [Refs. 9 and 10]. The mean and variance local estimates within a 5x5 window encompassing the selected pixel determine the value of k. The k value then specifies the new gray level of the pixel in question [Ref. 1:p. 24]. Virtual arrays again assist in the data manipulation and storage processes.

Kaeser's version of this filtering algorithm required that the user input the standard deviation of the image directly into the program [Ref. 4]. The operator had to obtain this value from the histogram function of the ImageAction software [Ref. 14:p. 11-9-3] before running the filter program. ImageAction*plus* software no longer provides the image mean and standard deviation calculations [Ref. 5]. Consequently, a new function, *dev(stddev)*, was devised to calculate the image standard deviation and input it directly into the filter routine without user intervention. This new function presents a vast improvement in terms of user convenience.

4. Program File *2sigma.c*

The 2sigma filter algorithm provides the basis for *2sigma.c* (Appendix D) and was adapted from the program introduced by Edwards [Ref. 1]. From the standard deviation of a Gaussian distribution, it is known that 95.5% of a selected region's pixels will be within two standard deviations of the local mean. An averaging formula has been devised in which pixels meeting the "2sigma" criteria are averaged for a given pixel, and those outside the range are rejected [Ref. 10]. These calculations are performed for 5x5 arrays of pixels. The program *2sigma.c* utilizes virtual arrays and the standard deviation function as well. It also has reduced size due to the removal of all general supporting functions into the program file *genfunc.c*.

5. Program File *threshit.c*

Threshit.c (Appendix E) produces a binary version of the filtered image. The program operator enters a threshold value which sets all pixel values above it to WHITE (level 255), and all those below to BLACK (level 0). The user then views the thresholded image and, if the result is not satisfactory, is given the option to enter a new threshold limit. *Threshit.c* utilizes a function named *threshit()* which is physically located within the *genfunc.c* program file. This organization was due to the fact that other program files, such as *feat_id.c*, call the *threshit()* function.

The key to the threshold process is a function call in the ITEX/PC*plus* software package called THRESHOLD. The ITEX/PC function is also named THRESHOLD and has the following format:

void threshold(*lowcut, highcut***);**

The *lowcut* and *highcut* integer values defined the lower and upper boundaries of the thresholded region. This threshold procedure could only be used on the currently

selected LUT [Ref. 12:p. 7-27]. The function in ITEX/PC*plus* software used to set a selected LUT to a prescribed threshold is:

void threshold(*group, bank, highcut, lowcut*);

where,

group specifies the LUT: RED, GREEN, BLUE, or INPUT,

bank specifies the output LUT used, in the range of 0 to 7,

highcut specifies the upper bound for the threshold, and

lowcut specifies the lower bound for the threshold.

The *threshit.c* program utilizes all four LUT groups and the 0*th* 256-byte bank to specify a linear transform output. This specification differs significantly from the original ITEX/PC function in the amount of versatility granted the programmer. Any of the LUTs can be modified with this threshold command, not just the current LUT [Ref. 6:p. 7-35]. The *lowcut* value must never be specified as higher than *highcut* value, but these values may be made equivalent to display one intensity at level 255 (WHITE).

Once the user is satisfied with a particular image threshold, the image can be mapped to memory. This procedure involves the use of another ITEX/PC*plus* function:

int maplut(*group, bank, x, y, dx, dy*);

MAPLUT modifies actual memory values rather than just LUT values as THRESH-OLD does. The *group* and *bank* variables are identical to those used in the THRESH-OLD function, but now only the INPUT LUT is modified for mapping purposes in

threshit.c. The quantities x, y, dx, and dy refer to the horizontal and vertical coordinates of the 512x512 image [Ref. 6:p. 7-17]. As in the case of the THRESHOLD function, ITEX/PC permits the user to transform only that area specified by the current LUT. The variables *group* and *bank* are not defined in ITEX/PC MAPLUT function [Ref. 12:p. 7-14].

6. Program File *feat_id.c*

It is mandatory that an image is thresholded prior to initiating the program *feat_id.c* (Appendix F). For this reason, the user is given the option within *feat_id.c* to threshold the image if thresholding was not completed previously. Conversion to a feature-labeled image requires an iterative process, whereby the image is evaluated row by row. Once this feature identification process is complete, the image can be further processed for specific feature data.

The *feat_id.c* program necessitated few modifications for compatibility with ITEX/PC*plus* software. Again, the more extensive changes were performed on the general support functions. *Feat_id.c* used the ITEX/PC*plus* functions WPIXEL and RPIXEL extensively, but these calls retained the same format as that used in ITEX/PC software [Refs. 6 and 12].

7. Program File *sizeit.c*

Program *sizeit.c* (Appendix G) creates an output file *size.dat* which contains the dimensions of all image features in a format necessary for input into a statistical analysis program. *Sizeit.c* relies upon pointer structures for dynamic memory allocation of processed feature data.

A conversion factor, SCALE_FACTOR, figures prominently in this sizing algorithm. Kaeser determined this factor to be 3.7353 contingent on image calibration [Ref. 4]. Upon subsequent runs of the *sizeit.c* program, however, a SCALE_FACTOR

of 10.0000 was necessary to achieve compatible results with Kaeser's program [Ref. 4:p. 17]. The constant term SCALE_FACTOR is defined in the header file *thesis.h* and can be easily modified if required. Based upon equipment configuration during hologram processing, the subject conversion factor is still questionable in terms of accuracy. For this reason, an additional function was devised to determine a conversion factor based upon user input. This function was entitled *scale()*, and its characteristics are discussed in the section relevant to *genfunc.c*.

8. Program File *speckle.c*

To calculate the value of speckle index (SI) after each filter iteration, *speckle.c* (Appendix H) is called to evaluate filtering effectiveness. This calculation is performed upon specified dimensions of the filtered image; integer values for image row and column are passed to this function from the filter routines. Values for speckle index typically range from a maximum value of 1.0, a rare occurrence, to a hypothetical value of 0 [Ref. 1:p. 20]. No program modifications were necessary for ITEX/PC*plus* compatibility. Speckle index is obtained purely from statistical manipulations.

9. Program File *vir_arry.c*

Vir_arry.c (Appendix I) consists of a group of functions supporting setup and access of the virtual disk-based array stratagem employed primarily by the filter routines. The virtual array functions were adapted by Kaeser [Ref. 4] from Reference 8. These arrays reside on disk and are accessed as though they are located in memory. This feature overcomes the memory limitation normally corresponding to array usage. Through the use of MACROS and pointer notation, the C programming language provides the means of managing virtual arrays [Ref. 8]. *Thesis.h* contains the obligatory MACROS needed for structure definition. All virtual array functions

mandated the use of long integers for indexing purposes [Ref. 13]. Since the dimensions of the actual image are defined in terms of integer values, a conversion interface problem developed in all the filter programs. This was further complicated by the fact that the ITEX/PC*plus* WPIXEL function references datum points in terms of integer values. This function is essential for "writing" the filtered pixel values to memory from the virtual array routines [Ref. 6:p. 5-4]. With the use of the C language "(long)" command, the integer indices referencing pixel values within the image can be used simultaneously to reference values within the virtual arrays with no conversion difficulty [Ref. 15:p. 36].

10. Program File *genfunc.c*

The program file *genfunc.c* (Appendix J) contains image processing general support functions. These functions maintain routine evolutions occurring within the main image processing programs. The following sections discuss functions that were either created or modified due to the advent of the ITEX/PC*plus* libraries.

a. Function *startit()*

This function makes provisions for initial Frame Grabber board setup. All the main function files with the exception of *speckle.c* and *vir_arry.c* access this function. Due to the introduction of the dual memory configuration, new ITEX/PC*plus* calls were required in addition to what Kaeser utilized [Ref. 4]. The SETDIM routine determines the configuration of the memory, either dual-store or single-store [Ref. 6:p. 2-6]. SELECT_MEM determines which frame memory will be accessed while in the dual-store mode. Frame memory choices consist of MEM_A or MEM_B as defined in the *itexpfg.h* header file [Ref. 6:p. 2-19]. DISPLAY_MEM must be used in conjunction with the SELECT_MEM routine. It specifies which memory area will be displayed and gives the user the option of writing to one image area while displaying

25

the other [Ref. 6:p. 2-20]. For the current application, MEM_A is both the accessed and displayed frame memory.

b. Function *readit()*

The image is read from disk storage and placed on the video monitor via the *readit()* function. Any comment associated with the image when it was stored will be displayed at that time. The ITEX/PC*plus* function READIM is the key to this procedure. The READIM call format is relatively unchanged from the ITEX/PC version. If an error occurs when a file is being read, the function will return an error code to identify why the operation failed. The *itexpfg.h* header file defines these error code values [Ref. 6:p. 6-5]. The standard ITEX/PC format did not make provisions for these values in its associated header file, *itexpc.h*; therefore, file errors are referred to numerically [Ref. 12:p. 6-7]. Accordingly, the *readit()* function was modified to reflect the error nomenclature.

c. Function *saveit()*

The *saveit()* function allows the user to save the image to disk memory via the ITEX/PC*plus* function SAVIM. The image can be stored using compression; although compression may, in rare cases, cause the image to occupy more disk space than an uncompressed version. If compression is used, the file will be automatically decompressed when read from the disk. As in the case of the READIM function, SAVIM returns error code values if the save operation is unsuccessful. Subsequent modifications were performed to the *saveit()* code to make it compatible with the new documentation [Refs 6 and 12].

d. Function *dev(stddev)*

Both program files *lstat.c* and *2sigma.c* require the computation of the standard deviation of the image for their respective filtering algorithms. The function

dev(stddev) computes this value through an iterative process utilizing the RPIXEL function. Since the standard deviation is computed directly from the image without user input, there is a slight increase in filter program execution times associated with the implementation of this function.

e.   Function *scale(factor)*

This function supports the *sizeit.c* program by allowing the user flexibility in the determination of a scale factor used in the sizing algorithm. The user is given the option of using the SCALE_FACTOR as delineated in the header file *thesis.h*, or of entering dimensions of a calibration object. If the user decides upon the latter, the length of the object must be entered in terms of pixels and inches. The function will then calculate the new SCALE_FACTOR and pass this value to the *sizeit.c* program.

E.   SUMMARY

The modifications and improvements performed on the image processing programs were necessary to make them compatible with the environment established by the installment of the PCVISION*plus* Frame Grabber board into the IBM PC/AT. These subject programs support all requirements for image acquisition, filtering, thresholding and particle identification. The following chapter examines the performance and verification of these programs at length.

IV. SOFTWARE PERFORMANCE ANALYSIS

This chapter will discuss the pertinent details concerning software performance. The subjects of executable program size, timing analysis, and program verification will be examined at length in the following sections. Any departures from results obtained from the original C language programs will be appraised.

A.   EXECUTABLE PROGRAM SIZE

Table 4.1 shows the comparison between executable program size of ITEX/PC and ITEX/PC*plus* supported software. The size information is presented in terms of kilobytes (KB) and was acquired via the use of the DOS command, DIR [Ref. 4]. The ITEX/PC-based programs written by Kaeser [Ref. 4] have significantly smaller executable file sizes. This is attributed to the fact that during the LINK process, Kaeser linked main programs to required library files only. It was not necessary to link to the object modules of the *genfunc.c*, *speckle.c*, or the *vir_arry.c* files since all necessary functions were appended to the end of the main program files [Ref. 4]. In addition, the ITEX/PC function library differs from the ITEX/PC*plus* version, and these differences will also impact executable file size.

TABLE 4.1: Comparisons of Executable Program Size (KB)

FILENAME	ITEX/PC	ITEX/PC*plus*
THRESHIT.EXE	31,467	63,101
FEAT_ID.EXE	32,509	64,407
SIZEIT.EXE	46,281	65,429
2SIGMA.EXE	49,483	65,491
LSTAT.EXE	49,475	65,499
GEOFIL.EXE	49,969	66.317

TABLE 4.2: Comparisons of Actual Program Size (KB)

FILENAME	ITEX/PC	ITEX/PC*plus*
THRESHIT.C	5,807	1,336
FEAT_ID.C	10,130	6,003
SIZEIT.C	11,011	7,426
2SIGMA.C	14,623	5,396
LSTAT.C	14,434	5,396
GEOFIL.C	14,888	7,916

One stated objective of this thesis was to optimize the existing C language code. Although executable program size comparisons contradict this goal, it is shown in Table 4.2 that actual program sizes for the ITEX/PC*plus* compatible programs are substantially reduced from the original versions. Two linker options, /F and /PAC, were initially employed to achieve more compact executable files. The use of these options should produce more rapid execution of files and shorter program load times [Ref. 16:p. 185]. The /E option had been utilized by Kaeser in Reference 4, but this file packing option may not always give satisfactory results. It can occasionally increase file size, and this obviously reduces savings in disk space [Ref. 17:p. 268].

Upon linking program files with the /PAC and /F options, it was found that these options did not pose an improvement over the /E option. In fact, executable file sizes were increased. For example, the *geofil.exe* file was 73,132 KB when these options were used in the LINK process. The /E option produced an executable file size of 66,317 KB. All other executable files were affected similarly. Accordingly, the /E option was utilized for the LINK procedure. Appendix K contains the listings of the batch files used to compile and link the program files.

TABLE 4.3: Program Execution Times

PROGRAM	TIME(SEC)
FEATURE IDENTIFICATION	809.82
FEATURE SIZING	1780.03
2SIGMA FILTER	760.55
LOCAL STATISTICS FILTER	1324.47
GEOMETRIC FILTER	3403.13

B. PROGRAM EXECUTION TIMES

The *j17res4.img* library image was used to test program execution times. This was the same image as that used by Kaeser in Reference 4, depicting an Air Force Resolution Target (AFRT) obscured by speckle noise. The unfiltered image *j17res4.ing* is depicted in Figure 4.1. Testing was accomplished by the use of a timer program, *timer.c*, in conjunction with a DOS input file to eliminate user intervention. Previous thesis work involved time testing of one quarter of the 512x512 image. MSFORT versions of the programs could effectively handle only one quadrant of the full screen image. Consequently, Kaeser conducted execution time comparisons on only one quarter of the image [Ref. 4]. Due to the banded nature of the ITEX/PC*plus* frame memory, quadrant timing tests of the image would not produce results that could be compared with the original findings [Ref. 5]; therefore, timing analysis of the programs was performed on the entire image. The resultant execution times are listed in Table 4.3.

C. PROGRAM VERIFICATION

To verify the results obtained from the ITEX/PC*plus*-based programs, two images were used from the existing image library. The *j17res4.img* was again used, this time to determine the accuracy of the filter programs. Both *feat_id.c* and *sizeit.c* were operationally tested with the image *10xwgrid.img* derived from a standard calibration
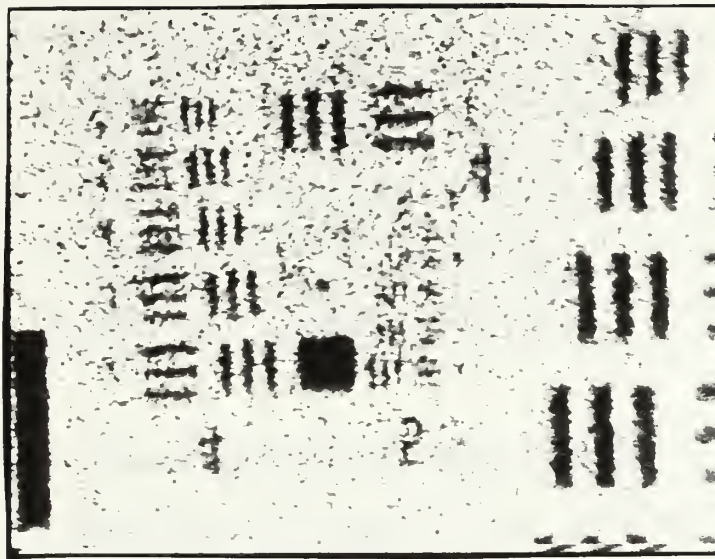
Figure 4.1: Unfiltered *j17res4.img.*

image [Ref. 2]. Resultant data acquired from these images was compared to data in Reference 4.

Speckle index value comparisons were used to determine the effectiveness of the filter programs. In each case, the respective filter was run through six iterations of the full screen image, *j17res4.img.* Table 4.4 contains the tabulation of the results acquired, and the results from the original C programs are listed in Table 4.5. A file comparison was performed to ensure that the image used for testing was identical to the full screen image used previously. Comparison of the data obtained from the current iterations to data from Kaeser's programs indicates slight variations in the speckle index values. The initial speckle index values for the two cases differ. One possible factor attributing to this difference is the inconsistency of integer conversions between the C language programs [Ref. 4]. The more significant measure of filter performance is the relative decrease in speckle index upon subsequent filter iterations.

31

TABLE 4.4: Speckle Index Reduction Results Obtained from Modified Programs

NUM	2 SIGMA	L STAT.	GEOMETRIC
0	0.293046	0.293046	0.293046
1	0.196282	0.180061	0.218982
2	0.144286	0.125157	0.174153
3	0.115754	0.099063	0.143760
4	0.098420	0.084035	0.121202
5	0.087068	0.074063	0.103756
6	0.079042	0.066919	0.089894

TABLE 4.5: Speckle Index Reduction Results (Ref. 4)

NUM	2 SIGMA	L STAT.	GEOMETRIC
0	0.304005	0.304005	0.304005
1	0.153767	0.194121	0.224875
2	0.098967	0.151951	0.178222
3	0.074415	0.129769	0.146152
4	0.060615	0.115938	0.122347
5	0.051691	0.106372	0.103983
6	0.045354	0.099367	0.089425

Therefore, the dissimilarities of initial speckle index value are not detrimental to filter function, and all filter programs produce desirable reduction of speckle. Consequently, all evidence indicates that the ITEX/P*Cplus*-based filter programs function correctly. Figure 4.2 illustrates *j17res4.img* after six iterations of the geometric filter.

The results obtained from the *feat_id.c* and *sizeit.c* programs were identical to those obtained from Kaeser's programs [Ref. 4]. The SCALE_FACTOR entry in the *thesis.h* header file had to be modified in order for the output files from the respective sizing programs to be congruent. Otherwise, the program files produced the same outputs as the original C language programs.
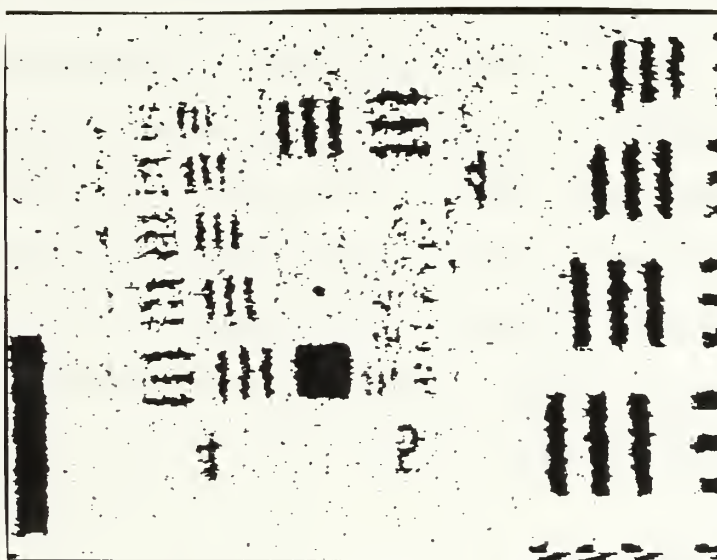
Figure 4.2: *j17res4.img* Image After Six Iterations of the Geometric Filter.

V. CONCLUSIONS

A. SUMMARY OF FINDINGS

This thesis accomplished all major objectives as set forth in the introductory chapter. No major problems were anticipated with the integration of the ITEX/PC*plus* libraries and the COMPAQ 386/20 computer system. Due to a memory conflict created within the COMPAQ, the PCVISION*plus* had to be installed into the IBM PC/AT to complete C program modifications. Therefore, the bulk of this research was conducted on the PC/AT used in previous work. Although several of the programs produced in Kaeser's research [Ref. 4] were modified with no major difficulties, type definitions created major faults in the filter routines which had to be resolved. It was the goal of this thesis to improve program portability and optimization. For the most part, this goal was achieved. Executable program size did increase, but this was the result of program structure changes and library modifications introduced by the ITEX/Pc*plus* system upgrade.

B. FUTURE WORK

Further verification testing is necessary to sufficiently demonstrate the operation of the C language programs. A minimal amount of experimentation was accomplished to compare the acquired results with data obtained from the previous version of the programs. Further testing is mandatory for complete program validation and should be pursued in subsequent thesis research prior to porting the programs over to other computer systems. Some effort was expended on the application of the NDP C-386 compiler to eliminate the need for virtual arrays for data processing. These efforts were largely unsuccessful due to the incompatibility of the ITEX/PC*plus* libraries

34

with the subject compiler system. This area of research could be further investigated for future thesis work. The following topics may also be considered for future work:

- Utilization of Macintosh board for image processing.

- Measurement of image particles utilizing a scanning electron microscope for sizing routine validation.

- Further validation of image measurements using new calibration objects.

- Integration of SUN workstations for image processing.

APPENDIX A: PROGRAM HEADER FILE:
thesis.h

```
/*  PURPOSE:
        To completely list all necessary include files, manifest constants,
    and MACRO definitions required by all the programs within this
    package.  Any changes to constants in thesis.h will guarantee
    redefinition of those parameters throughout the main files.

 */


/*      Include files for use with ITEXPC programs
 */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <malloc.h>
#include <graph.h>
#include <conio.h>
#include "itexpfg.h"
#include "stdtyp.h"



/*   Program constants used in the C program modules of the ITEXPC
        Optical System used for Rocket motor Hologram analysis.   */

/*  Intial ITEXPC Board Jumper Settings   */

#define  MEMBASE    0xD0000L   /* Itex board base memory start address */
#define  REGBASE    0x300      /* Itex board base register start address */
#define  MEMORY     DUAL       /* Itex board memory type */

/* ITEXPC initial AOI (area of interest) settings  */

#define IXS        0
#define IYS        0
#define NROW       480
#define NCOL       512
#define COLOR      150
#define XSIZE      512
#define YSIZE      512
#define DEPTH      8

/*  ItexPC LUT (Look Up Table) Variables       */
```

```c
/*  Threshold Limits  */
#define  LOWEST   0             /* Equates to Black for lowcut value */
#define  HIGHEST  255           /* Equates to White for highcut value */
#define  BLACK    0
#define  WHITE    255


/* Filter array sizes and limits  */

#define  SNUM     25
#define  NUM      9
#define  HIGH     254
#define  LOW      0


/* Sizing magnification factor used in sizing program */

#define  SCALE_FACTOR  10.000000   /* Based on ( 317.5 / 85 )   */

/* Virtual array Header File  Definitions    */
#define  header         7

/* Virtual Array Control Block typedef  */

typedef struct {

    FILE *file;              /* pointer to file control block */
    long size;               /* number or array elements in file */
    int elsize;              /* number or bytes in each element */
    char *buffer;            /* pointer to array buffer */
    int buf_elsize;          /* size of element in buffer including index */
    int buf_size;            /* number of elements in buffer */
    char *blank_rec;         /* pointer to initialization record */
    /* used for extending file */
}
VACB ;                   /* Virtual Array control block type name */


/* Virtual Array Access Prototypes */

int init_v_array(char *filename,int rec_size,char filchar);
VACB *open_v_array(char *filename,int buffer_size);
void close_v_array(VACB *v_array);
void *access_v_rec(VACB *v_array,long index);

/* Virtual Array Access Macros */

#define VREC(y) ((items *)access_v_rec(item_array,y))
#define gdesc(y) VREC(y)->v_gdesc
#define G(x,y) VREC(y)->v_gdesc[x]

/* Virtual Array element structure typedef */
```

```
typedef struct {
    unsigned char v_gdesc[NCOL];
}
items;
```

APPENDIX B: PROGRAM FILE: geofil.c

```
/*PURPOSE   : Provides for filtering an image using the geometric filter
              algorithm.  This program processes the image and provides
              for the operator to select the number of iterations.

                  The area of filtering for the image is controlled by the
              operator by keyboard input of number of ROWS and COLS to
              process.  Program will allow for a Maximum input of 480 rows
              and 512 cols with a minimum of 1 row  and 1 column.

              The resulting filtered image pixel value is stored on
              disk in a virtual array, that is  written to the image
              processing screen when all calculations are complete.

              Total calculation time for a 512x480 image is 3403.13 sec.

              DISK SPACE REQUIRED for virtual array storage is 250 KB minimum
              available on the disk at start of the routine.

 */
#include "thesis.h"

char    filename[20], comline[200];
int     srow, scol, times, row, col;
main()

{
     int          flag;
     extern int   row, col, srow, scol, times;

     printf("\n\tReady to Load IMAGE FILE?...Yes (y) / No (n) ");
     flag=getch();
     if(flag == 'Y'|| flag == 'y')
     {
          startit();
          readit();
     }
     if( flag ==  'N' || flag ==  'n' ) return(0);

       geti("\n\nEnter number of rows (Max=480) to use in SPECKLE CALC : ",&srow);
       geti("\nEnter number of columns (Max=512) to us in SPECKLE CALC :  ",&scol);
       geti("\n\n\nEnter number of filter iterations to do...",&times);
       geti("\nEnter the number of IMAGE rows (MAX = 480) to FILTER : ",&row);
       geti("\nEnter the number of IMAGE cols (MAX = 512) to FILTER : ",&col);
```

```c
    switch(times)
    {
        case  0:
            speckle();
            printf("\nSave image to Disk File?..Yes (y) / No (n) ");
            flag=getch();
            if(flag == 'Y'|| flag == 'y') saveit();
            exit(1);
            break;

        case  1:
            break;

        default:
            break;
    }

    do
     {
       speckle();
       geofil();
    }
    while(times == 0);
    printf("\nSave image to Disk File?..Yes (y) / No (n) ");
    flag=getch();
    if(flag == 'Y'|| flag == 'y') saveit();
}


/*  geometric filter algorithm  */

geofil()
{
    VACB            *item_array;
    extern int      row, col, times;
    int             f1, f2, g1, g2, g3, x, y;
    int             a, b, c, d, pixel, calls;

    /* create a virtual array for the array of filtered
       pixel values the size of the image ( 480x512 )    */

    init_v_array("ITEMS.VAR",sizeof(items),NULL);

    /* open the virtual array, reserve buffer space for 100 elements */

    item_array = open_v_array("ITEMS.VAR",100);

    /* start local statistic filter routine   */

    if( row > NROW ) row = NROW;
    if( col > NCOL ) col = NCOL;
```

```c
for( y = 1 ; y < row ; y++ )          /* Zero the initial array  */
{
  for( x = 1 ; x < col ; x++ )
      {
         G(((long)x),((long)y)) = NULL;
      }
}
for(calls = times; calls != 0; calls--)
{
    c = 4;
    printf("\n\n\tFilter Running --> %d Runs after this one!...\n",calls-1);
    for( ; c ; )
    {
        switch(c)
        {
        case 4 :

            a=1;  b=0;  c=3;  d=1;
            printf("\nc  = 3 ");
            t1(a,b,c,d,item_array);
            break;

        case   3:

            a=0;
            b=1;
            c=2;
            printf("\nc  = 3 ");
            t1(a,b,c,d,item_array);
            break;

        case   2:

            a=1;
            b=1;
            c=1;
            printf("\nc  = 2 ");
            t1(a,b,c,d,item_array);
            break;

        case    1:

            a=1;
            b=-1;
            c=0;
            printf("\nc  = 1 ");
            t1(a,b,c,d,item_array);
            printf("\nc = 0 END");
            speckle();
            break;
```

41

```
                default   : break;
                }
            }
        }
        /* close the virtual array */
        close_v_array(item_array);
    }



/* subroutine for geofil for steps 1 and 2 */

    t1(a,b,c,d,item_array)
        int     a, b, c, d;
        VACB    *item_array;
    {
            int           f1, f2, g1, g2, g3;
            int           pixel, x, y;
            long          x1, y1;
            extern int    row,col,times;

            for( y = 1 ; y < row-1 ; y++ )
            {
                for( x = 1 ; x < col-1 ; x++ )
                {
                    x1 = ((long) x);
                    y1 = ((long) y);
                    f1 = rpixel( x,y );
                    f2 = rpixel( x-a,y-b );
                    G(x1,y1)= max( f1, min( f2-1, f1+1 ) );
                }
            }
            printf("  step 1");
            for( y = 1 ; y < row-1 ; y++ )
            {
                for( x = 1 ; x < col-1 ; x++ )
                {
                    x1 = ((long) x);
                    y1 = ((long) y);
                    g1 =  G( x1,y1);
                    g2 =  G(((long)(x-a)),((long)(y-b)));
                    g3 =  G(((long)(x+a)),((long)(y+b)));
                    pixel = max( g1, min( g2, min( g1+1, g3+1 ) ) );
                    wpixel(x,y,pixel);
                }
            }
            printf("  step 2");
            if( d==1 )
            {
                    a= -a;
                    b= -b;
```

42

```
                        d=  0;
                        t1(a,b,c,d,item_array);
                }
                else if( d==0 )
                {
                        d=1;
                        t2(a,b,c,d,item_array);
                        return;
                }
        }


/* subroutine for geofil for steps 3 and 4 */
   t2(a,b,c,d,item_array)
     int             a, b, c, d;
     VACB            *item_array;

   {
        int             f1, f2, g1, g2, g3;
        int             x, y, pixel;
        long            x1, y1;
        extern int      row, col, times;

        for( y = 1 ; y < row-1 ; y++ )
        {
            for( x = 1 ; x < col-1 ; x++ )
            {
                x1 = ((long) x);
                y1 = ((long) y);
                f1 = rpixel( x,y );
                f2 = rpixel( x-a,y-b );
                G(x1,y1)= min( f1, max( f2-1, f1+1 ) );
            }
        }
        printf("  step 3");
        for( y = 1 ; y < row-1 ; y++ )
        {
                for( x = 1 ; x < col-1 ; x++ )
                {
                    x1 = ((long) x);
                    y1 = ((long) y);
                    g1 =  G( x1,y1);
                    g2 =  G(((long)(x-a)),((long)(y-b)));
                    g3 =  G(((long)(x+a)),((long)(y+b)));
                    pixel = max( g1, min( g2, min( g1+1, g3+1 ) ) );
                    wpixel(x,y,pixel);
                }
        }
        printf("  step 4");
        if( d==1 )
        {
```

```
                a= -a;
                b= -b;
                d=  0;
                t2(a,b,c,d,item_array);
        }
        else if( d==0 )
        {
                d = 1;
                return;
        }
}
```

APPENDIX C: PROGRAM FILE: lstat.c

```
/*PURPOSE  : Provides for filtering an image using the local statistical
            algorithm.  This program processes the image by using a local
            5x5 array of pixels to calculate a statistical value for the
            central pixel of the local array.

                The area of filtering for the image is controlled by the
            operator by keyboard input of number of ROWS and COLS to
            process.  Program will allow for a Maximum input of 480 rows
            and 512 cols with a minimum of 1 row  and 1 column.  Program
            also requires the image Std. Dev. as calculated by the genfunc.c
            Program.  The resulting filtered image pixel value is stored on
            disk in a virtual array, that is  written to the image processing
            screen when all calculations are complete.

                Total calculation time for a 512x480 image is 1324.47 sec.

                DISK SPACE REQUIRED for virtual array storage is 250 FB
            minimum available on the disk at the start of the routine.
*/

#include "thesis.h"

char    filename[20], comline[200];
int     row, col, srow, scol, times;
float   dev1;

main()

{
        int             flag;
        extern int      row, col, times;

        printf("\n\tReady to Load IMAGE FILE?...Yes (y) / No (n) ");
        flag=getch();
        if(flag == 'Y'|| flag == 'y')
        {
                startit();
                readit();
        }
        if(flag == 'N' || flag == 'n') return(0);

        geti("\n\nEnter number of rows (Max=480) to use in SPECKLE CALC : ",&srow);
        geti("\nEnter number of columns (Max=512) to us in SPECKLE CALC :  ",&scol);
        geti("\n\n\nEnter number of filter iterations to do...",&times);
```

45

```
         geti("\nEnter the number of IMAGE rows (MAX = 480) to FILTER : ",&row);
         geti("\nEnter the number of IMAGE cols (MAX = 512) to FILTER : ",&col);
         dev(&dev1);
         printf("\nThe standard deviation for this image is  %.2f\n",dev1);


         do
         {
             speckle();
             lstat();
         }
             while(times == 0);
             printf("\nSave image to Disk File?..Yes (y) / No (n) ");
             flag=getch();
             if(flag == 'Y'|| flag == 'y') saveit();
     }

/*.....................................................................*/

lstat()

{
         VACB            *item_array;
         long            x1,y1;
         extern float    dev1;
         extern int      row, col, times, srow, scol;
         register int    m, n, nn;
         int             pixel,a,b,x,y,calls;
         float           sum1, sum2, svar, lvar, tvar;
         float           lmean, lmean2, stddev, stddev2, k;
         int             ldata[SNUM];   /* SNUM defined in Header file  */

         /* create a virtual array for the array of filtered
            pixel values the size of the image ( 480x512 )     */

         init_v_array("ITEMS.VAR",sizeof(items),NULL);

         /* open the virtual array, reserve buffer space for 100 elements */

         item_array = open_v_array("ITEMS.VAR",100);

         /* start local statistic filter routine    */

         if( row > NROW ) row = NROW;
         if( col > NCOL ) col = NCOL;

         stddev = dev1 / 255.0 ;              /* Normalize deviation value */
         stddev2 = stddev * stddev ;

         for(calls = times; calls != 0; calls--)  {
             printf("\n\n\tFilter Running --> %d Runs after this one!...\n",calls-1);
```

46

```
for ( y = 2 ; y < row - 2 ; y++)
{
     for ( x = 2 ; x < col - 2 ; x++)
     {
          pixel = rpixel( x,y );
          sum1 = 0;
          sum2 = 0;
          nn = 0;

          for( n = y - 2 ; n < y + 3 ; n++ )
          {
               for( m = x - 2 ; m < x + 3 ; m++ )
               {
                    ldata[nn] = rpixel( m,n );
                    sum1 += ldata[nn];
                    nn++;
               }
          }

          lmean =  sum1 / SNUM  ;
          lmean2 = lmean * lmean ;
          for( nn = 0 ; nn < SNUM ; nn++ )
          {
               svar = (ldata[nn] - lmean ) * ( ldata[nn] - lmean );
               sum2 += svar;
          }
          lvar = sum2 / SNUM ;
          tvar = fabs( ( (lvar + lmean2) / (stddev2 + 1) ) - lmean2 ) ;
          k =  tvar / ( ( stddev2 * lmean2 ) + tvar ) ;
          x1 = ((long) x);
          y1 = ((long) y);
          G( x1,y1 ) = (int) ( lmean + k * ( pixel - lmean ) ) ;
     }
}

/* write contents of the G array to image screen */

printf("\n\a\tWriting filtered image to screen...");

for( b = 2 ; b < row - 2 ; b++ )
{
     for( a = 2 ; a < col - 2 ; a++ )
     {
          wpixel( a,b,G(((long)a),((long)b)));
     }
}
speckle();
}
/* close the virtual array */
```

```
        close_v_array(item_array);
}
```

APPENDIX D: PROGRAM FILE: 2sigma.c

```
/*PURPOSE   : Provides for filtering an image using a 2-sigma statistical
              algorithm.  This program processes the image by using a local
              5x5 array of pixels to calculate a statistical value for the
              central pixel of the local array.  The algorithm will eliminate
              pixels from the summation that are greater than 2-Sigma in
              value from that of the central pixel.

                  The area of filtering for the image is controlled by the
              operator by keyboard input of number of ROWS and COLS to
              process.  Program will allow for a Maximum input of 480 rows
              and 512 cols with a Minimum of 1 row  and 1 column.  Program
              also requires the image Standard deviation as obtained using
              genfunc.c program.  The resulting filtered pixel value is
              stored on disk in a virtual array and written to screen later.

                  Total calculation time for a 512x480 image is 760.55 sec

                  DISK SPACE REQUIRED for virtual array storage is 250 KB
              minimum available on the disk at start of the routine.

 */
#include "thesis.h"

char          filename[20], comline[200];
int           row, col,t imes, srow, scol;
float         dev1;

main()

{
     int          flag;
     extern int   row, col, times;

     printf("\n\tReady to Load IMAGE FILE from disk?...Yes (y) / No (n) ");
     flag=getch();
     if(flag == 'Y'|| flag == 'y')
     {
          startit();
          readit();
     }
     if(flag == 'N' || flag == 'n') return(0);
     geti("\n\nEnter number of rows (Max=480) to use in SPECKLE CALC : ",&srow);
     geti("\nEnter number of columns (Max=512) to us in SPECKLE CALC :   ",&scol);
     geti("\n\n\nEnter number of filter iterations to do...",&times);
```

49

```c
        geti("\nEnter the number of IMAGE rows (MAX = 480) to FILTER : ",&row);
        geti("\nEnter the number of IMAGE cols (MAX = 512) to FILTER : ",&col);
        dev(&dev1);
        printf("\n\tThe standard deviation for this image is:  %.2f\n",dev1);
        do
            {
             speckle();
             sigma();
            }
        while( times == 0 );

        printf("\nSave image to Disk File?..Yes (y) / No (n) ");
        flag=getch();
        if(flag == 'Y'|| flag == 'y') saveit();

}

/******************************************************************/

sigma()
{
        VACB            *item_array;
        extern int      row, col, times, srow, scol;
        extern float    dev1;
        long            x1, y1;
        register int    m, n, nn;
        int             pixel, sum1, sum2, delta, high;
        int             low, a, b, x, y, calls;
        float           hvar, lvar, stddev;
        int             ldata[SNUM];            /* SNUM defined in Header file  */

        /* create a virtual array for the array of filtered
           pixel values the size of the image ( 512x480 )          */

        init_v_array("ITEMS.VAR",sizeof(items),NULL);

        /* open the virtual array, reserve buffer space for 100 elements */

        item_array = open_v_array("ITEMS.VAR",100);

        /* start local statistic filter routine    */

        if( row > NROW ) row = NROW;
        if( col > NCOL ) col = NCOL;

        stddev = dev1 / 255.0 ;             /* Normalize deviation value */
        hvar = 1. + 2. * stddev ;        /* find high and low sigma   */
        lvar = 1. - 2. * stddev ;

        for (calls = times; calls != 0; calls--)  {
            printf("\n\n\tFilter running --> %d Runs after this one!...\n",calls-1);
```

50

```
    for ( y = 2 ; y < row - 2 ; y++)
    {
        for ( x = 2 ; x < col - 2 ; x++)
        {
            pixel = rpixel( x,y );
            sum1 = 0;
            sum2 = 0;
            nn = 0;
            delta = 0;
            high = (int)(hvar * pixel);
            low = (int)(lvar * pixel);

            for( n = y - 2 ; n < y + 3 ; n++ )
            {
                for( m = x - 2 ; m < x + 3 ; m++ )
                {
                    x1 = ((long) x);
                    y1 = ((long) y);
                    ldata[nn] = rpixel( m,n );
                    if( ( low <= ldata[nn] ) && ( ldata[nn] <= high ) )
                    {
                        sum1 += ldata[nn];
                        delta++;
                    }
                    nn++;
                }
            }

            if(delta <= 2)     /* correct shot noise -- 4 neighbor average */
            {
                sum2 = ( rpixel(x,y-1)+rpixel(x,y+1)+rpixel(x-1,y)+rpixel(x+1,y) );
                G( x1,y1 ) = (pixel + sum2) / 5;
                continue;
            }

            G( x1,y1 )  =  sum1 / delta  ;
        }
    }

    printf("\n\tWriting filtered image to screen...");

    for( b = 2 ; b < row - 2 ; b++ )
    {
        for( a = 2 ; a < col - 2 ; a++ )
            wpixel( a,b,G(((long)a),((long)b)) );
    }
    speckle();
}
    /* close the virtual array */
```

51

```
        close_v_array(item_array);
}
```

APPENDIX E: PROGRAM FILE: threshit.c

```
/*PURPOSE   : This program thresholds the image on screen by taking
              an operator input value and forcing all image pixel
              values above the threshold value to the value of BLACK
              and all those below the value to the WHITE value.

              ( Background == WHITE ; Feature == BLACK )


 */
#include "thesis.h"
char          filename[20], comline[200];
int           srow, scol;

main()
{
     int flag;

     printf("\nThis program will threshold the input image desired.");
     printf("\n\n\tReady to Load Image?...Yes (y) / No (n) ");
     flag=getch();
     if(flag == 'Y'|| flag == 'y')
     {
          startit();
          readit();
     }

     if(flag == 'N'|| flag == 'n')
     {
          initialize();
          return(0);
     }

     printf("\n\n\tTHRESHOLD the image?...Yes (y) / No (n) ");
     flag = getch();
     if(flag == 'Y'|| flag == 'y') threshit();

}
```

APPENDIX F: PROGRAM HEADER FILE:
feat_id.c

```c
/*PURPOSE   : Labels and identifies each feature in an image on the ITEX
              system.  Reads pixel-by-pixel and groups the objects by
              assigning a unique ID number to each feature or object so
              that they can be processed by other routines.

                 The ID routine requires a thresholded on image screen image
              or input of a previously saved thresholded image from disk or
              imput from disk and then thresholding. This program module
              includes an optional call to threshold if desired.

                 ( Background == WHITE ; Feature == BLACK )

 */
#include "thesis.h"

char    filename[20], comline[200];
int     srow, scol;

main()
{
     int flag;

     printf("\n\aYou MUST USE A THRESHOLDED IMAGE for this Program!!");
     printf("\n\n\tReady to Load image from disk?...Yes (y) / No (n) ");
     flag=getch();
     if(flag == 'Y'|| flag == 'y')
     {
          startit();
          readit();
     }
     if(flag ==  'N' || flag ==  'n' ) return(0);

     printf("\n\n\tNeed to THRESHOLD the image?...Yes (y) / No (n) ");
     flag = getch();
     if(flag == 'Y'|| flag == 'y') thresh();

     feat_id();

     printf("\n\nSave image to Disk File?..Yes (y) / No (n) ");
     flag=getch();
     if(flag == 'Y'|| flag == 'y') saveit();
}
```

54

```
/*  image feature identification algorithm  */

feat_id()
{
     register int    x, y;
     int             a, x1, y1, x2, y2, x3, y3, x4, y4;
     int             fid = 0;
     int             gid = 0;
     int             temp_fid = 0;
     int             temp_gid = 0;
     int             maxfl, n1, n2, n1a, n1b, n1c, kk, kk1, nn1;

     printf("\n\n\tSTEP ONE in Progress...");

     for( y = 1 ; y < NROW ; y++ )
     {
         for( x = 0 ; x < NCOL ; x++ )
         {
             if( rpixel(x,y) == WHITE ) continue;
             if ( x == 0 )
             {
                 wpixel( x,y,fid );
                 continue;
             }
             if( ( rpixel( x-1,y ) != WHITE ) )
             {
                 wpixel( x,y,rpixel(x-1,y) );
                 continue;
             }
             if( rpixel(x,y-1) != WHITE )
             {
                 wpixel( x,y,rpixel( x,y-1 ) );
                 continue;
             }
             wpixel( x,y,fid );
             a = x + 1;

             while(1)
             {
                 if( rpixel( a,y ) == WHITE )
                 {
                     wpixel( x,y,fid );
                     step( &fid,&gid );
                     break;
                 }
                 if( rpixel(a,y-1) != WHITE )
                 {
                     wpixel( x,y,rpixel( a,y-1 ) );
                     break;
                 }
```

55

```
                        a += 1;
                }
        }   /* end x - for */
}       /* end y - for */

fid += gid*HIGH;

printf("\n\n\tSTART STEP TWO... ");
printf("\n\nEnter the BEST estimate of the max feature length");
geti("\n\nIt's Better to be too large than too small!!",&maxfl);
printf("\n\n\tRUNNING");

for( y = 1 ; y < NROW ; y++ )
{
        for( x = 1 ; x < NCOL ; x++ )
        {
                n1 = rpixel( x,y );
                n2 = rpixel( x,y-1 );
                if( ( n1 || n2 == WHITE ) || (n1 == n2 )) continue;

                x3 = x - maxfl;
                x4 = x + maxfl;
                y3 = y - maxfl;
                y4 = y + maxfl;

                if( x3 < LOW) x3 = LOW;
                if( x4 > NCOL ) x4 = NCOL;
                if( y3 < LOW) y3 = LOW;
                if( y4 > NROW ) y4 = NROW;

                for ( y1 = y3 ; y1 < y4 ; y1++ )
                {
                        for ( x1 = x3 ; x1 < x4 ; x1++)
                        {
                                n1a = rpixel( x1,y1 );
                                if ( n1a == WHITE) continue;
                                if ( n1a == n1 ) wpixel( x1,y1,n2 );
                        }
                        fid -= 1;
                }
        }
}

printf("\n\n\tSTEP THREE in Progress");

for ( y = 1 ; y < NROW ; y++ )
{
        for( x = 1 ; x < NCOL ; x++ )
        {
                kk = rpixel( x,y );
                if( kk == WHITE ) continue;
```

56

```
            kk1 = kk;
            nn1 = temp_fid + temp_gid *  HIGH;
            if( kk1 < nn1 ) continue;
            if( (kk1 - nn1) < -350 ) kk1 = kk + 508;
            if( (kk1 - nn1) >= -350  && (kk1 - nn1) < -150 )
                 kk1 = kk + 254;
            if( kk1 != nn1 )
            {
                 x3 = x - maxfl;
                 x4 = x + maxfl;
                 y4 = y + maxfl;

                 if( x3 < 0) x3 = LOW;
                 if( x4 > NCOL-1 ) x4 = NCOL -1;
                 if( y4 > NROW-1 ) y4 = NROW -1;

                 for( y1 = y ; y1 < y4 ; y1++ )
                 {
                     for( x1 = x3 ; x1 < x4 ; x1++)
                     {
                         n1c = rpixel( x1,y1 );
                         if( ( n1c == WHITE ) || ( n1c != kk ) )continue;
                         wpixel( x1,y1,temp_fid  );
                     }
                 }
                 step( &temp_fid,&temp_gid );
            }
        }
    }

    printf("\n\n\t\tFEATURE COUNT IS:   %d",fid);
    maplut( INPUT, 0, IXS, IYS, NCOL, NROW );
}
```

APPENDIX G: PROGRAM FILE: sizeit.c

```c
/*PURPOSE  :  This routine uses an existing or saved image that has been
             thresholded and then ID'd with the ID Algorithm.  The output
             of this program is a tabular output of the calculated Area,
             X-chord, and Y-chord that is suitable for input to a
             Statistical Analysis Program.  Output file is written to
             default drive as SIZE.DAT.

             ( Background == WHITE ; Feature == BLACK )

 */

#include "thesis.h"

char        filename[20], comline[200];
int         srow, scol;

main()
{
     int flag;

     printf("\n\aUSE IMAGE processed by the FEATURE ID program!!");

     printf("\n\tReady to Load ID'd image from disk?...Yes (y) / No (n) ");
     flag=getch();
     if(flag == 'Y'|| flag == 'y')
     {
          startit();
          readit();
     }
     if( flag ==  'N' || flag ==  'n' ) return(0);
     initialize();
     size_it();

     printf("\n\tSave image to Disk File?...Yes (y) / No (n)  ");
     flag=getch();
     if(flag == 'Y'|| flag == 'y') saveit();

}

/*  image feature sizing algorithm  */

size_it()
{
     register int     nn, x, y, xa, ya, xb, yb;
     int              j, ip, ipa, ipb, x1, y1, x2, y2, maxfl, num;
```

58

```c
int             xmax, ymax, xmin, ymin, maxarea, minarea;
int             xflag=0;
int             yflag=0;
int             aflag=0;
float           area, xchord, ychord, factor;
float           fxmax, fymax, fxmin, fymin, fmaxarea, fminarea;
float           cf_1, cf_2, mag;
long            *aptr;
long            *xptr;
long            *yptr;
long            *stop;
FILE            *featdata;

geti("\nEnter the number of Features",&num);
getf("\nEnter Magnification Factor",&mag);
geti("\nEnter the Maximum Feature Length",&maxfl);

/* Set initial test min/max values  */

xmax =     0;
ymax =     0;
maxarea = 0;
xmin =     512;
ymin =     512;
minarea = 512;

/* Dynamically allocate memory space for the number of
   features entered by the operator.

       Note: This eliminates the need to declare large arrays
       for these values and allows maximum number of features      */

xptr = ( long *) calloc( num ,sizeof(long));
yptr = ( long *) calloc( num ,sizeof(long));
aptr = ( long *) calloc( num ,sizeof(long));

/* Check for successfull memory allocation    */

if( !xptr || !yptr || !aptr )
{
    printf("\nOut of Memory!!! \n");
    printf("\nYou may have entered too many features.\n");
    printf("\nCorrect problem and try again.");
    exit(0);
}

/* Calculate Magnification Constants    */

scale(&factor);
cf_1 = ( factor / mag );
cf_2 = ( cf_1 * cf_1 );
```

```c
/* Above Conversion factor converts size to microns
        and is related to the input magnification for the
        hologram.  The defined " SCALE_FACTOR " constant
        value must be set for the proper value based on the
        equipment set up during actual image aquisition.
        The constant term SCALE_FACTOR is defined in the
        Header File THESIS.H
 */

/* Begin Sizing Routine   */

printf("\n\n\tStarting to SIZE!...");

nn = 0;             /* Feature number counter  */

for( y = 0 ; y < NROW ; y++ )
{
     if( nn == num )break;        /* Quit when all features sized */
     for( x = 0 ; x < NCOL ; x++ )
     {
          ip = rpixel(x,y);
          if( (ip == WHITE) || (ip == BLACK) )continue;
          if( nn == num )break;   /* Quit when all features sized */

          x1 = x - maxfl;                    /* Set up sizing Box  */
          y1 = y;
          x2 = x + maxfl;
          y2 = y + maxfl;

          if( x1 < LOW)  x1 = LOW;
          if( y1 < LOW)  y1 = LOW;
          if( x2 > NCOL )  x2 = NCOL;
          if( y2 > NROW )  y2 = NROW;

          for( ya = y1 ; ya < y2 ; ya++ )
          {
               for( xa = x1 ; xa < x2 ; xa++ )
               {
                    ipa = rpixel(xa,ya);
                    if( ipa != ip )continue;
                    aflag++;
                    xflag++;
               }
               if( xflag >  *(xptr + nn ) ) *(xptr + nn) = xflag;
               xflag = 0;
          }
          *(aptr + nn) = aflag;
          aflag = 0;
```

```
            for( xb = x1 ; xb < x2 ; xb++)
            {
                for( yb = y1 ; yb < y2 ; yb++ )
                {
                    ipb = rpixel(xb,yb);
                    if( ipb != ip ) continue;
                    yflag++;
                    wpixel(xb,yb,BLACK);
                }
                if( yflag > *(yptr + nn ) ) *(yptr + nn) = yflag;
                yflag = 0;
            }

            /* Calculate Min/Max values  */

            xmax = max( xmax, *(xptr + nn) );
            ymax = max( ymax, *(yptr + nn) );
            maxarea = max( maxarea, *(aptr + nn) ) ;
            xmin = min( xmin, *(xptr + nn) ) ;
            ymin = min( ymin, *(yptr + nn) ) ;
            minarea = min( minarea, *(aptr + nn) ) ;

            nn++;                   /* increment counter  */
            printf("\n\n\tFeature %d complete...",nn);
        }
}      /*.... end  initial x/y  for loop ....*/

/* output data section ·  */

printf("\n\a Sending output data to screen and FILE SIZE.DAT !");
/* wait();   */

if( ( featdata = fopen("size.dat","w") ) == NULL )
{
    printf("CANNOT Open output file %s\n", *featdata);
    return(1);
}

printf("    ID NO      AREA    X-Width   Y-Width \n");

for( j = 0 ; j < num ; j++ )
{
    fprintf( featdata,"%10d %10.3f %10.3f %10.3f\n", j+1, *(aptr + j) * cf_2,
    *(xptr + j) * cf_1, *(yptr + j) * cf_1 );

    printf("\n%10d %10.3f %10.3f %10.3f ", j+1, *(aptr + j) * cf_2,
    *(xptr + j) * cf_1, *(yptr + j) * cf_1 );
}

/* This section can be used to put total information on bottom of data file
   if desired.  File written now to be used with STATGRAPHICS            */
```

61

```
/*      fprintf(featdata,"\n\n\nMax X-chord= %f    Max Y-chord= %f    Max Area= %f ",
        xmax*cf_1,ymax*cf_1,maxarea*cf_2);
        fprintf(featdata,"\nMin X-chord= %f    Min Y-chord= %f    Min Area= %f ",
        xmin*cf_1,ymin*cf_1,minarea*cf_2);  */

printf("\n\n\nMax X-chord= %f    Max Y-chord= %f    Max Area= %f ",
xmax*cf_1,ymax*cf_1,maxarea*cf_2);
printf("\nMin X-chord= %f    Min Y-chord= %f    Min Area= %f ",
xmin*cf_1,ymin*cf_1,minarea*cf_2);

/* Close open file and free allocated memory   */

fclose(featdata);
free(xptr);
free(yptr);
free(aptr);
}
```

APPENDIX H: PROGRAM FILE: speckle.c

```
/*PURPOSE   : Provides for calculating  the  speckle index of an image.
              The resulting value is used as a measure to evaluate the
              sucess of filtering an image to reduce the speckle noise.
              This routine is called in all filter algorithms.

        --> OPERATOR must enter a Number of ROWS and COLUMNS
            THE PROGRAM uses to calclate the speckle index.

                  ROWS --->  Maximum = 480      Minimum = 1
                  COLS --->  Maximum = 512      Minimum = 1

 */
#include "thesis.h"
speckle()
{
      extern int        srow, scol;
      int               m, n, x, y, nn;
      long              smax, smin;
      long              sdata[NUM];

      unsigned long deviation, ssum;

      float   smean, slocal, stotal, tot;
      float   spklindex;

      if( srow >= NROW ) srow = NROW;
      if( scol >= NCOL ) scol = NCOL;

      tot = (long)srow * scol;

      /* Commence calculation   */

      printf("\n\n\tCalculating Speckle INDEX...");

      for ( n = 1 ; n < srow-2 ; n++)
      {
          for ( m = 1 ; m < scol-2 ; m++)
          {
                smax = 0;
                smin = 260;
                ssum = 0;
                nn = 0;
```

```c
            for( y = ( n-1 ) ; y < (n+2) ; y++ )
            {
                for( x = ( m-1 ) ; x < ( m+2 ) ; x++)
                {
                    sdata[nn] = rpixel( x,y );
                    ssum += (long)sdata[nn];
                    if( smax < sdata[nn] ) smax = sdata[nn];
                    if( smin > sdata[nn] ) smin = sdata[nn];
                    nn++;
                }
            }
            deviation = smax - smin;
            if( ssum == 0 ) smean = 1;
            smean = ssum / NUM   ;
            slocal = (float)deviation / smean;
            stotal += slocal;
        }
    }
    spklindex = stotal / tot;
    printf("\n\n\tThe Calculated %d by %d speckle index is %f ",srow,scol,spklindex);
}

/*.......... END Speckle Function ................................*/
```

APPENDIX I: PROGRAM FILE: vir_arry.c

```
/*PURPOSE  : These functions provide for the setup of a disk drive virtual
             array that can be indexed as if it were in the calling
             programs data storage area.  The routines provide for disk
             file access when required to retrieve data elements from the
             array. This allows the size of a declared array to be limited
             only by the amount of DISK SPACE available. See "thesis.h"
             header file for setup and MACRO routines that support these
             functions.  Also program lstat.c fully implements the array
             routines.

 */
#include "thesis.h"

/* Virtual Array Access Routines */

int       init_v_array(filename,rec_size,filchar)
char      *filename, filchar;
int       rec_size;

{
    long    size;
    FILE    *f;
    f = fopen(filename,"wb");
    if (f != NULL)
    {
        size = 0;
        fwrite(&size,4,1,f);            /* write array size of zero */
        fwrite(&rec_size,2,1,f);        /* and array element size   */
        fwrite(&filchar,1,1,f);         /* and fill char            */
        fclose(f);                      /* to file header */
        return(1);
    }
    else
        return(NULL);
}

VACB      *open_v_array(filename,buffer_size)
char      *filename;
int       buffer_size;
{
    VACB    *v_array;
    char    *buf_ptr;
    int     i;
    char    filchar;
```

65

```
/* allocate virtual array control block */

v_array = (VACB *) malloc(sizeof(VACB));
if (v_array == NULL) return(NULL);

/* open virtual array file */

v_array->file = fopen(filename,"r+b");
if (v_array->file == NULL)
{
    free(v_array);
    return(NULL);
};

/* get array size and element size for control block */

fread(&v_array->size,4,1,v_array->file);
fread(&v_array->elsize,2,1,v_array->file);
fread(&filchar,1,1,v_array->file);
v_array->buf_elsize = v_array->elsize + 4;

/* allocate buffer */


v_array->buffer = (char *) malloc(v_array->buf_elsize * (buffer_size + 1));
if (v_array->buffer == NULL)
{
    fclose(v_array->file);
    free(v_array);
    return(NULL);
};
v_array->buf_size = buffer_size;

/* set up blank_rec using the fill character in array header */
/* for initializing new array elements */

buf_ptr = v_array->buffer + v_array->buf_elsize * v_array->buf_size;
v_array->blank_rec = buf_ptr + 4;
for (i = 0; i < v_array->buf_elsize; i++)
    *buf_ptr++ = filchar;

/* set record index negative for all buffer elements */

buf_ptr = v_array->buffer;
for (i = 0; i < v_array->buf_size; i++)
{
    *((long *)buf_ptr) = -1L;
    buf_ptr += v_array->buf_elsize;
};
return(v_array);
```

```
}

void            close_v_array(v_array)
VACB            *v_array;
{
     int    i;
     char   *buf_ptr;
     long   rec_index, file_offset;

     buf_ptr = v_array->buffer;


     /* flush buffer */

     for (i = 0; i < v_array->buf_size; i++)
     {

          /* check each element index */
          /* if element present; write it to disk */

          rec_index = *((long *)buf_ptr);
          if (rec_index >= 0)
          {
               file_offset = header + rec_index * v_array->elsize;
               fseek(v_array->file,file_offset,0);
               fwrite(buf_ptr + 4, v_array->elsize,1, v_array->file);
          };
          buf_ptr += v_array->buf_elsize;
     };
     free(v_array->buffer);                    /* de-allocate buffer */
     fclose(v_array->file);                    /* close array file   */
     free(v_array);                            /* de-allocate VACB   */
}


void            *access_v_rec(v_array,index)
VACB            *v_array;
long index;
{

     char    *buf_ptr;
     int     buf_index;
     long rec_index, temp_index;


     /* calculate buffer address of referenced element */

     buf_index = index % v_array->buf_size;
     buf_ptr = v_array->buffer + buf_index * v_array->buf_elsize;
     rec_index = *(long *)buf_ptr;
```

```c
        /* if element present, return its buffer address */
        if (rec_index == index) return(buf_ptr + 4);

        /* if element doesn't exist, extend the file */

        if (index >= v_array->size)
        {
            fseek(v_array->file,0,2);
            for (temp_index = v_array->size; temp_index++ <= index; )
                fwrite(v_array->blank_rec, v_array->elsize, 1, v_array->file);
            v_array->size = index + 1;
            fseek(v_array->file,0,0);
            fwrite(&v_array->size, 4, 1, v_array->file);
        };

        /* if buffer slot is occupied by another element, */
        /* save it to disk  */

        if (rec_index >= 0)
        {
            fseek(v_array->file, rec_index * v_array->elsize + header, 0);
            fwrite(buf_ptr + 4, v_array->elsize, 1, v_array->file);
        };

        /* read referenced element into buffer slot */

        fseek(v_array->file, index * v_array->elsize + header, 0);
        fread(buf_ptr + 4, v_array->elsize, 1, v_array->file);
        *((long *)buf_ptr) = index;

        /* return address of element in buffer */

        return(buf_ptr + 4);
}

/*................................................................*/
```

APPENDIX J: PROGRAM FILE: genfunc.c

```c
/*PURPOSE  : These general functions provide for routine evolutions
            that occur a number of times in the Image processing
            routines developed for analysis of speckle reduction
            algorithms. Some require support of other functions
            and are not totally independant.

 */
#include "thesis.h"
/* initial ITEXPC Board setup  */

startit()
{
    sethdw(REGBASE,MEMBASE,MEMORY);
    setdim(XSIZE,YSIZE,DEPTH);
    aclear(IXS,IYS,NCOL,NROW,COLOR);
    fgon();
    initialize();
    cls();
    select_mem(MEM_A);
    display_mem(MEM_A);
}

/* Get integer keyboard input  */

geti(name,iptr)
char   *name;
int    *iptr;
{
    printf(" %s ",name);
    scanf(" %d",iptr);
}


/* Get floating point keyboard input  */

getf(rname,riptr)
char   *rname;
int    *riptr;
{
    printf("%s ",rname);
    scanf(" %f",riptr);
}


/* Pause routine used to prompt for operator intervention*/
```

```c
wait()
{
     printf("\n\n Press Return to continue");
     fflush(stdin);
     getchar();
}


/* returns minimum integer value from input array passed */

getmin(data,num)

int    *data;
int    num;
{
     int t, min;

     for (min= data[0], t=1 ; t<num ; t++)
          if( data[t] < min ) min = data[t];

     return min;
}

/* returns maximum integer value from input array passed */

getmax(data,num)

int *data;
int num;
{
     int t, max;

     for (max= data[0], t=1 ; t<num ; t++)
          if( data[t] > max ) max = data[t];

     return max;
}
/*  DOS System Call to clear Display Screen  */

cls()
{
     _clearscreen(_GCLEARSCREEN);
}


/*  routine to read an image from disk  */

readit()
/*program uses the following externals: filename,comline*/
{
     extern char   filename[20], comline[200];
```

70

```c
        int             errval;
        int             ch;

        while(1)
        {
            printf("\n\nENTER IMAGE FILENAME as (DEV:name.IMG)->");
            fflush(stdin);
            scanf("%s",filename);
            errval = readim(IXS,IYS,NCOL,NROW,filename,comline);

            if(errval == 0)  {
                printf("\nFILE-> %s \n\nCOMMENT: %s",filename,comline);
                break;
            }
            if(errval != 0)  {
                switch(errval)
                {
                    case FILE_ERROR:
                        printf("Error opening file\n");
                        break;
                    case FORMAT_ERROR:
                        printf("Unknown file format\n");
                        break;
                    case READ_ERROR:
                        printf("Error Reading file\n");
                        break;
                    default:
                        printf("? Unknown Error ?%d\n",errval);
                        break;
                }
                printf("Try Again?? ...Yes(y) / No(n)\n");
                ch = getch();
                if(ch == 'Y' || ch == 'y') continue;
                if(ch == 'N' || ch == 'n') exit(0);
            }
        }
}


/*  routine to save an image to disk  */

saveit()
/*program uses the following externals: *filename,*comline*/
{
        extern char  filename[20], comline[200];
        int             format, errval;
        int             ch;

        initialize();
        while(1)
        {
```

```c
        printf("\n\nENTER FILENAME...(DEV:name.IMG)->");
        fflush(stdin);
        scanf("%s",filename);
        geti("\n Enter File Compression Value ?...(0/1)",&format);
        printf("\n\n Enter Image Comment ..to continue.. (Max 200 CHAR)...");
        fflush(stdin);
        gets(comline);

        if(format == 1)printf("\n\nStoring Image Using Compression -- Please wait !!");
        if(format == 0)printf("\nStoring Image -- Please wait!!");

        errval = saveim(IXS,IYS,NCOL,NROW,format,filename,comline);

        if(errval == 0)  {
            printf("\nImage save completed Sat");
            break;
        }

        if(errval != 0)  {
            printf("\n\a Error saving file!!");
            if(errval == ALLOCATION_ERROR) {
                printf("\nInsufficient Disk Space");
            }
            if(errval == WRITE_ERROR) {
                printf("\nError writing file or values");
            }
            printf("\n\tTry Again?? ...Yes(y) / No(n)");
            ch = getch();
            if(ch == 'Y' || ch == 'y') continue;
            if(ch == 'N' || ch == 'n') break;
        }
    }
}

/* counter step routine */

step(u,v)
int *u,*v;
{
    *u += 1;
    if(*u >= HIGH)  {
        *u = 1;
        *v += 1;
    }
}

/*  threshold routine for feat_id.c*/

thresh()
{
    int          ans,c,option;
```

72

```
        unsigned int  val;

        while(1)
        {
            initluts();
            initialize();
            geti("\n\n\tENTER NEW THRESHOLD LEVEL (0-255) :",&val);
            setlut(INPUT,0);
            threshold(INPUT,0,HIGHEST,val);
            setlut(RED,0);
            threshold(RED,0,HIGHEST,val);
            setlut(GREEN,0);
            threshold(GREEN,0,HIGHEST,val);
            setlut(BLUE,0);
            threshold(BLUE,0,HIGHEST,val);
            setlut(INPUT,0);

            printf("\n\nSATISFIED WITH THRESHOLD AT %d?...Yes(y) / No(n)...",val);
            ans = getch();
            if(ans == 'Y' || ans == 'y') break;
        }
        maplut(INPUT,0,IXS,IYS,NCOL,NROW);
        initialize();
}

/*   threshold routine for threshit.c   */

threshit()
{
    int    ans,c,option;
    int    *val;

    while(1)
    {
        initluts();
        initialize();
        geti("\n\nENTER NEW THRESHOLD LEVEL (0-255) :",&val);
        printf("%d\n",val);
        setlut(INPUT,0);
        threshold(INPUT,0,HIGHEST,val);
        setlut(RED,0);
        threshold(RED,0,HIGHEST,val);
        setlut(GREEN,0);
        threshold(GREEN,0,HIGHEST,val);
        setlut(BLUE,0);
        threshold(BLUE,0,HIGHEST,val);
        setlut(INPUT,0);

        printf("\n\nSATISFIED WITH THRESHOLD AT %d?...Yes(y) / No(n)...",val);

        ans = getch();
```

```
            if(ans == 'Y' || ans == 'y') break;
    }
    printf("\n\t\t 0: SAVE THE THRESHOLD IMAGE?...");
    printf("\n\t\t 1: QUIT (Leave thresholded image for further processing)...");
    printf("\n\t\t 2: RESTORE system to original input image...");
    geti("\n\t Select option by NUMBER: ",&option);
    switch(option)
    {
        case 0:
            maplut(INPUT,0,IXS,IYS,NCOL,NROW);
            saveit();
            break;
        case 1:
            maplut(INPUT,0,IXS,IYS,NCOL,NROW);
            break;
        case 2:
            initialize();          /*initialize without mapping operation*/
            break;
        default:
            break;
    }
}


/*   function to compute standard deviation of image    */

dev(stddev)
   float       *stddev;
{
   float        var, sum, sqsum, sqvalue, dim, value;
   long         x, y;

   sum = sqsum = 0;
   dim = (float)NROW * NCOL;
   for( y = 0 ; y <=  NROW-1 ; y++)
   {
      for( x = 0 ; x <=  NCOL-1 ; x++)
      {
         value = rpixel(x,y);
         sqvalue = value * value;
         sum += value/dim;
         sqsum +=  sqvalue/dim;
      }
   }

   var = sqsum - (sum*sum);
   *stddev =  sqrt(var) ;
}



/* function to compute  scale factor for sizing routine   */
```

```
scale(factor)
    float           *factor;


{

    int             inch, pixel, flag;
    float           dinch, dpixel;      /* default length values */
    /* dinch and dpixel are to be modified accordingly */
    dinch  =  1.00;
    dpixel = 50.00;

    printf("\n\tUtilizing standard SCALE_FACTOR?...Yes (y) / No (n)");
    flag = getch();
    if(flag == 'Y' || flag == 'y')
    {
        *factor = ((float)SCALE_FACTOR);
    }

    if(flag == 'N' || flag == 'n')
    {
        printf("\n\tPress '0' if using default values for length in inches or pixels");
        geti("\n\tEnter the length of object in inches(default):",&inch);
        geti("\n\tEnter the length of object in pixels(default):",&pixel);

            if(inch == 0)
            {
                inch = dinch;
            }
            if(pixel == 0)
            {
                pixel = dpixel;
            }
        *factor = ((float)inch) / ((float)pixel);
    }
    printf("\n\tSCALE_FACTOR is %f\n",*factor);
}
```

APPENDIX K: C COMPILER OPERATIONS

The following batch files list the C compiler directives used to compile and link all C programs. The C compiler files ACCOMP.BAT and DACCOMP.BAT are listed below. ACCOMP.BAT contains the switches necessary to utilize CODEVIEW debugger, and ACCOMP2.BAT is for final program optimization. The F drive listed in the PATH statement of these batch files is due to the fact that the programs were stored on a Bernoulli disk.

```
ACCOMP.BAT:
path c:\;c:\dos;c:\wsjet;f:\kedit.*;c:\mouse1;c:\xtree;c:\lsrctrl;...
    f:\;f:\msc\bin;f:\msc\tmp;f:\msc\src;f:\msc\wrk;f:\msc\lib;...
    f:\msc\include
set include=\msc\include
set lib=\msc\lib;\pcplus\itex\lib
set tmp=\msc\tmp

cl /c /Fs /Zi /G0 /AL /Fm %1.c

ACCOMP2.BAT
path c:\;c:\dos;c:\wsjet;f:\kedit.*;c:\mouse1;c:\xtree;c:\lsrctrl;...
    f:\;f:\msc\bin;f:\msc\tmp;f:\msc\src;f:\msc\wrk;f:\msc\lib;...
    f:\msc\include
set include=\msc\include
set lib=\msc\lib;\pcplus\itex\lib
set tmp=\msc\tmp

cl /c /Fs /Ze /G0 /AL  %1.c
```

The following batch files, LTHESIS.BAT and DTHESIS.BAT, link the programs to the libraries and supporting function files. The *speckle.c* should not be included within these files when linking the programs *sizeit.c* and *feat_id.c* due to an undefined external problem associated with the external variables SROW and SCOL. LTHE-SIS.BAT is used for debugging applications, and DTHESIS.BAT is for final program

optimization. It is also unnecessary to link *feat_id.c* and *sizeit.c* to the virtual array functions contained within *vir_arry.c.*

```
LTHESIS.BAT:
IF NOT ERRORLEVEL 1 LINK /NOD /CO /E %1 genfunc speckle vir_arry ...
                ,,,ITEXPCML.LIB LLIBCE.LIB;
```

```
DTHESIS.BAT:
IF NOT ERRORLEVEL 1 LINK /NOD /E %1 genfunc speckle vir_arry...
                ,,,ITEXPCML.LIB LLIBCE.LIB;
```

To create the menu format for program display, a C program named *hologram.c* was generated. To link all image processing programs to this module, all program "main()" statements were modified so that *hologram.c* was made the main program. All image processing programs were then linked to *hologram.c* after compilation. The ACCOMP2.BAT file was used for compilation, and the PTHESIS.BAT file was used for subsequent module linking. Due to the length of the LINK statement, the user must enter the libraries ITEXPCML and LLIBCE manually into the keyboard each time the LINK process is accomplished.

```
PTHESIS.BAT:
IF NOT ERRORLEVEL 1 LINK /NOD /E  %1 table speckle vir_arry...
                genfunc threshit sizeit feat_id geofil..
                1stat 2sigma
```

REFERENCES

1. Edwards, T. D., *Implementation of Three Speckle Reduction Filters for Solid Propellant Combustion Holograms*, Master's Thesis, Naval Postgraduate School, Monterey, California, December 1986.

2. Redman, D. N., *Image Analysis of Solid Propellant Combustion Holograms Using an Imageaction Software Package*, Master's Thesis, Naval Postgraduate School, Monterey, California, June 1986.

3. Orguc, E. S., *Automatic Data Retrieval from Rocket Motor Holograms*, Master's Thesis, Naval Postgraduate School, Monterey, California, December 1987.

4. Kaeser, D. S., *Code Optimization of Speckle Reduction Algorithms for Image Processing of Rocket Motor Holograms*, Master's Thesis, Naval Postgraduate School, Monterey, California, December 1988.

5. Imaging Technology Incorporated, *ImageActionplus User's Guide*, Imaging Technology Incorporated, April 1987.

6. Imaging Technology Incorporated, *ITEX/PCplus Programmer's Manual*, Imaging Technology Incorporated, April 1987.

7. Imaging Technology Incorporated, *PCVISIONplus Frame Grabber User's Manual*, Imaging Technology Incorporated, April 1987.

8. Imaging Technology Incorporated, *PCVISION Frame Grabber Manual*, Imaging Technology Incorporated, July 1985.

9. Crimmins, T. R., *Geometric Filter for Reducing Speckle*, Optical Engineering, v. 25, pp. 651-654, May 1986.

10. Lee, J. S., *Speckle Suppression and Analysis for Synthetic Aperture Radar*, Optical Engineering, v. 25, pp. 636-643, May 1986.

11. Microsoft Corporation, *Microsoft Quick C Toolkit*, v. 2.0, Microsoft Press, 1988.

12. Imaging Technology Incorporated, *The ITEX/PC Programmer's Manual*, Imaging Technology Incorporated, 1985.

13. Tichenor, Mark, "Virtual Arrays in C," *Dr. Dobb's Journal of Software Tools*, #138, pp. 46-66, May 1988.

14. Imaging Technology Incorporated, *The ImageAction User's Guide*, Imaging Technology Incorporated, 1985.

15. Kernighan, Brian W., and Ritchie, Dennis M., *The C Programming Language*, Second Edition, Prentice Hall, 1988.

16. Microsoft Corporation, *Microsoft C 5.0 Optimizing Compiler User's Guide*, Microsoft Corporation, 1987.

17. Microsoft Corporation, *Microsoft C 5.0 Optimizing Compiler Microsoft Codeview and Utilities*, Microsoft Corporation, 1987.

INITIAL DISTRIBUTION LIST